# A DESIGN METHODOLOGY FOR OPTOELECTRONIC VLSI

by

Richard Gregory Rozier III

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering

Charlotte

2007

Approved by:

______________________________

Dr. Fouad E. Kiamilev

______________________________

Dr. Thomas P. Weldon

______________________________

Dr. Dian Zhou

______________________________

Dr. Hassan M. Razavi

______________________________

Dr. Phillip E. Johnson

| Report Documentation Page | | | | | | *Form Approved* *OMB No. 0704-0188* |
|---|---|---|---|---|---|---|
| Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. | | | | | | |
| 1. REPORT DATE **2007** | | 2. REPORT TYPE | | | | 3. DATES COVERED **00-00-2007 to 00-00-2007** |
| 4. TITLE AND SUBTITLE **A Design Methodology for Optoelectronic VLSI** | | | | | | 5a. CONTRACT NUMBER |
| | | | | | | 5b. GRANT NUMBER |
| | | | | | | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | | | | | | 5d. PROJECT NUMBER |
| | | | | | | 5e. TASK NUMBER |
| | | | | | | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of North Carolina at Charlotte,Department of Electrical and Computer Engineering ,Charlotte,NC,28223** | | | | | | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | | | | | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | | | | | | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |
| 12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited** | | | | | | |
| 13. SUPPLEMENTARY NOTES | | | | | | |
| 14. ABSTRACT **see report** | | | | | | |
| 15. SUBJECT TERMS | | | | | | |
| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON | |
| a. REPORT **unclassified** | b. ABSTRACT **unclassified** | c. THIS PAGE **unclassified** | **Same as Report (SAR)** | **160** | | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

# ABSTRACT

RICHARD GREGORY ROZIER III. A Design Methodology for Optoelectronic VLSI
(Under the direction of DR. FOUAD E. KIAMILEV)

A methodology for designing silicon complementary metal-oxide semiconductor (CMOS) integrated circuits (ICs) for implementation with optoelectronic (OE) technology is presented. Optoelectronic technology involves the bonding of laser detectors and transmitters to ports on the surface of the silicon IC. The ports that connect to the laser devices are metal pads that are placed in a gridded area array. The methodology focuses only on the design of the digital logic circuitry of an FSOI system and consists of partitioning the design at the system level for a multiple IC implementation, using the Very high-speed integrated circuit Hardware Description Language (VHDL) as a means of design entry and definition, using computer-aided design (CAD) tools for logic synthesis and automatic placement and routing of transistors, and enhancing the CAD tools to perform automated placement and routing of the area array pads. The results of applying this approach to a number of ICs are discussed also.

# DEDICATION

The quest to obtaining a Ph.D. is a long and difficult journey. No one does it alone. There is always a foundation of love, support, and encouragement to help the student succeed. For me, that foundation is my family -- my mother, sister, and brother-in-law. I would like to thank them for everything they have given me along the way.

# ACKNOWLEDGEMENTS

I would like to recognize Dr. Fouad Kiamilev as one of the strongest and most positive influences in my life. He’s the one who gave me a chance to do what I really wanted to do. Were it not for him, I wouldn’t be where I am today.

I would like to thank the members of my examining committee -- Dr. Thomas Weldon, Dr. Dian Zhou, Dr. Hassan Razavi, and Dr. Phillip Johnson. I would like to thank Mr. Ray Farbarik of Duet Technologies for supplying and supporting the area-pad routing software, Eggo. I would like to express my gratitude to Dr. Ashok Krishnamoorthy of Lucent Technologies for all the friendship, support, and opportunities he has given me. I would like to acknowledge Jeremy Ekman, Premanand Chandramani, Jim Rorie, Jim Rieve, and Rodney Dyer as colleagues, consultants, proofreaders, and most importantly, friends who helped me keep my sanity. Finally, I would like to express my special thanks to Hilah Cannon, who kept me in line and inspired me during one of the biggest challenges of my life.

Portions of this work were supported by a grant from the Air Force Office of Scientific Research.

# TABLE OF CONTENTS

LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1 - INTRODUCTION

The integrated circuit (IC) electronics industry is changing rapidly. IC processing technology is advancing such that the minimum feature size for a complementary metal-oxide semiconductor (CMOS) silicon transistor is shrinking, resulting in increased transistor density and improved performance. Leading-edge silicon computer chips contain more than 5 million transistors per chip and operate at frequencies greater than 300MHz. As data processing abilities have improved with smaller and faster ICs, chip-to-chip communication has started to evolve also. Conventional means of chip-to-chip communication, such as perimeter wire-bonded ICs in a ceramic or plastic package that are soldered to a copper-clad printed circuit (PC) board, are no longer sufficient for today's high-speed ICs. A processing chip that can compute data at a rate of 500Mbits/sec is underutilized on a PC board that allows only 100Mbits/sec of bandwidth. To overcome this communication bottleneck, chip-to-chip connection schemes with fewer limitations have been explored. One such scheme that holds a lot of promise is free-space optical interconnects (FSOI). FSOI has two major advantages over the conventional PC board approach. First is that it replaces the slower, all-metal connection from chip-to-chip with a faster laser light source and laser light detector that are attached to the surface of the chip such that data is communicated in the form of laser light pulses. Therefore, the signal is not subject to the resistive, capacitive, and inductive parasitics that the PC board has, and can switch at a higher frequency. Optical circuits have been shown to operate at frequencies greater than 500 Mbits/

sec [1]. The second benefit of FSOI is that the laser sources and detectors are placed in a gridded array on the surface of the chip, instead of wire bonding just around the perimeter. This allows for a much greater number of inputs and outputs (I/Os) to be accessed on a single chip -- on the order of thousands instead of hundreds. Systems have been demonstrated with 4096 I/Os [2].

With the changes in the IC technology comes a change in the very large-scale integration (VLSI) design approach. A new design methodology has to be adopted to take advantage of the benefits that FSOI offers. Optoelectronic VLSI is the coupling of optical devices, that can detect and transmit laser light, with silicon CMOS VLSI circuitry. A new methodology for optoelectronic VLSI is the subject of this dissertation. This procedure has been developed as the result of more than four years of designing custom ICs whose sizes range from a few thousand transistors to 180,000 transistors, and whose functions range from a simple first-in, first-out memory circuit to an intelligent 16-channel self-routing data packet switch. It makes use of hardware description languages (HDL) as the means of design entry and definition, as well as high-level computer-aided design (CAD) tools that perform automated placement and routing (APR) of CMOS transistors. This new method, shown as a flow chart in Figure 1-1, includes partitioning of the design at the system level, integrating mixed-signal analog and digital circuits, and enhancing the existing CAD tools to perform placement and routing of the area-array pad circuitry.

## 1.1 Objectives

Because the IC electronics industry is changing and progressing so quickly, there is a constant need to update and improve the engineering and design process to more efficiently and effectively produce results. The design methodology described in this disserta-

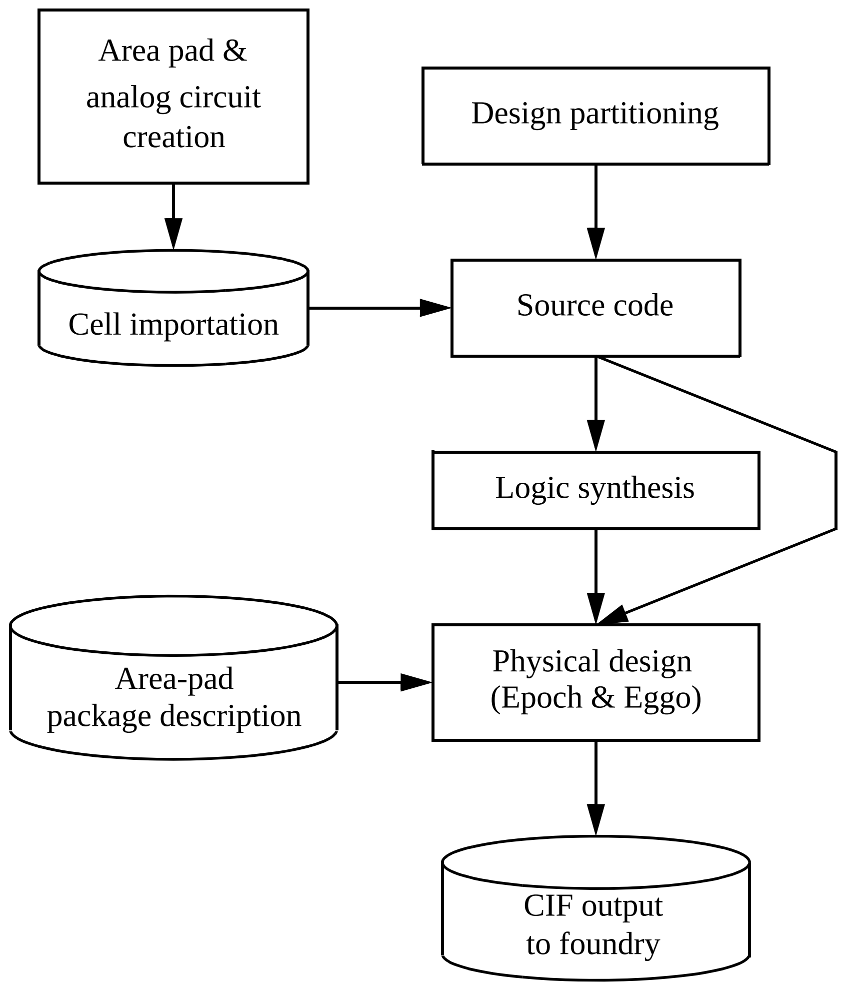


Figure 1-1. Flow chart of design methodology

tion seeks to maintain the efficiency of VLSI design by addressing the evolutionary changes and automating the design flow as much as possible. It also attempts to provide a top-down design strategy that is robust enough to handle future changes in the technology without having to be significantly modified.

1.2 Dissertation organization

Chapter 2 provides some background information on FSOI to provide some context and help the reader understand the motivation for some of the steps in the design process.

The partitioning of a project into logical and physical blocks from the design specifications is discussed in Chapter 3.

Chapter 4 covers the setup and preparation of the area pads and the importation of any analog or special circuitry.

Source code creation and CAD tool execution is examined in Chapter 5.

The results and conclusions of applying this method to the design of a number of ICs are discussed in Chapter 6.

# CHAPTER 2 - BACKGROUND INFORMATION

## 2.1 Conventional chip-to-chip communication

### 2.1.1 IC packaging and printed circuit board disadvantages

The need for an improvement in chip-to-chip communication is now greater than ever. Digital Equipment Corporation has announced plans for a 1GHz version of their Alpha processor [3]. That chip will probably be packaged and placed on a PC board that will only allow a communication bandwidth of 100MHz. The effectiveness of this chip in a computer system will be determined by its data throughput, where the throughput is a performance specification such as the number of instructions processed per second. While a fast chip may boast to be able to process a very large number of instructions per second, if the data cannot be transferred to the chip at that same rate, then either the chip will sit idle through many clock cycles waiting for data to arrive or the processed data will have to be buffered while it is waiting to leave the chip. This performance disparity is the result of the parasitic effects of the layers of interconnect that the signal travels through on its way from IC to IC. These parasitics are caused not only by the soldering of the chip onto a copper-clad PC board, but also by bonding of the die in a plastic or ceramic package.

Figure 2-1 shows the conventional means of packaging an IC. The die is epoxied in the center of the cavity of a ceramic or plastic package. All the data signals, as well as power and ground, are bonded from metal pads around the perimeter of the die to the fingers of the package with very small wires usually made of gold. The fingers of the pack-

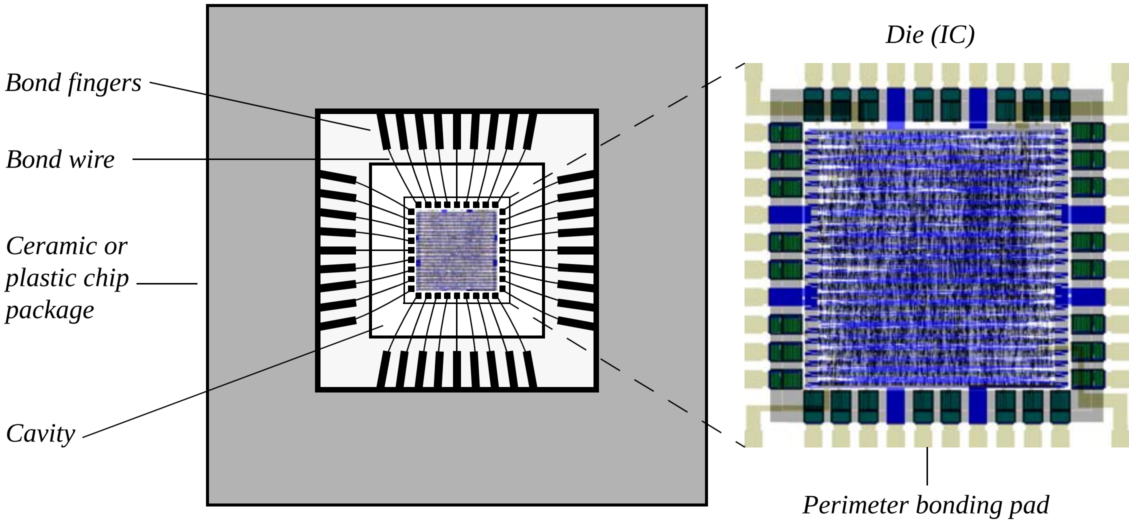


Figure 2-1. Conventional IC packaging

age are also made of metal, usually gold, and they connect to the pins of the package through metal traces. Once the packaging process is finished, the result is called a “chip” and is then placed and soldered to a PC board. Other chips are also soldered to the PC board and connections are made from chip to chip via traces of copper wire, as shown in Figure 2-2. The signal from a logic gate on one chip to a logic gate on another chip

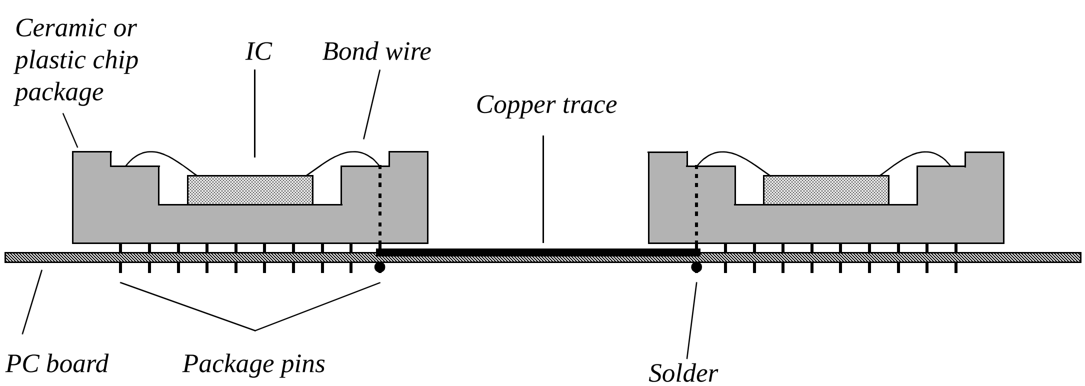


Figure 2-2. PC board chip-to-chip communication

through a buffer at the output port of the chip. It then passes through a long series of metal connections as follows:

1. The bond wire (chip 1)
2. The bond finger (chip 1)
3. The package pin (chip 1)
4. The solder and copper trace
5. The package pin (chip 2)
6. The bond finger (chip 2)
7. The bond wire (chip 2)

Once the signal reaches the bond wire of the second chip, then it travels through the bond pad and on its way to the input of the logic gate. All the metal that carries the signal has a lot of capacitance (C) and some resistance (R). These contribute to a large RC time constant ($\tau$ = RC), which is the amount of time required for a capacitive network to be charged to a certain voltage. Larger values of R and C will lead to longer charging times, which in turn leads to a lower switching frequency (f), since $f \sim 1/\tau$ . In addition, the bond wires and the package pins have some inductance (L). The voltage across an inductive circuit is directly proportional to the change in the current through the circuit over time as shown in Equation 2-1.

$$v_L = L\frac{di_L}{dt} = L\frac{\Delta i_L}{\Delta t} \qquad \text{(EQ. 2-1)}$$

$$f = \frac{1}{\Delta t} \qquad \text{(EQ. 2-2)}$$

$$v_L = L\Delta i_L f \qquad \text{(EQ. 2-3)}$$

Therefore, as frequency increases so will the voltage drop ($v_L$) across the bond wires and the pins. As a result, the voltage at the input of a logic gate whose signal comes from off-chip might not be enough to make the gate switch. Similarly, a gate that sends an output signal to another chip might lose voltage across the wire and pin of its own package and not be able to drive the input on another chip. In summary, it can be seen that the limitations in the switching frequency of the I/O of a chip on a PC board can be attributed to the cumulative resistance, capacitance, and inductance of the chip’s package as well as the solder and copper connection of the PC board.

### 2.1.2 Perimeter wire bonding

There is another drawback to the standard method of packaging an IC for use in a PC-board system. Wire bonding around the perimeter of the IC, as shown in Figure 2-1, doesn’t take advantage of the entire area that could be available for I/O communication. Since metal-oxide semiconductor field-effect transistor (MOSFET) integration in silicon is a three-dimensional process, there is room for connections to be made on the top layer of metal over active logic circuitry that is ordinarily unutilized. This limitation in the number of I/Os per chip combined with the inherent switching constraints of an all-metal chip-to-chip connection restricts the bandwidth over which chips can communicate with each other. Thus, there is a communication bottleneck between chips.

## 2.2 Emerging IC communication schemes

A number of alternative chip-to-chip connection strategies have been explored. Among them, multi-chip modules and free-space optical interconnects stand out as providing improvements over traditional PC board implementations.

### 2.2.1 Multi-chip modules

In order to overcome the problems presented by conventional chip-to-chip communication, various solutions have been proposed. One that is receiving a lot of attention is multi-chip modules (MCMs). MCMs have a variety of interconnect schemes but all types consist of placing a number of ICs in close proximity (less than 1cm) so that they can operate in the same large package. The interconnect strategies vary from wire bonding on the perimeter bond pads of one IC directly to the perimeter bond pads of another IC, as in Figure 2-3, to flip-chip solder bump bonding the ICs onto a multi-layer metal substrate that

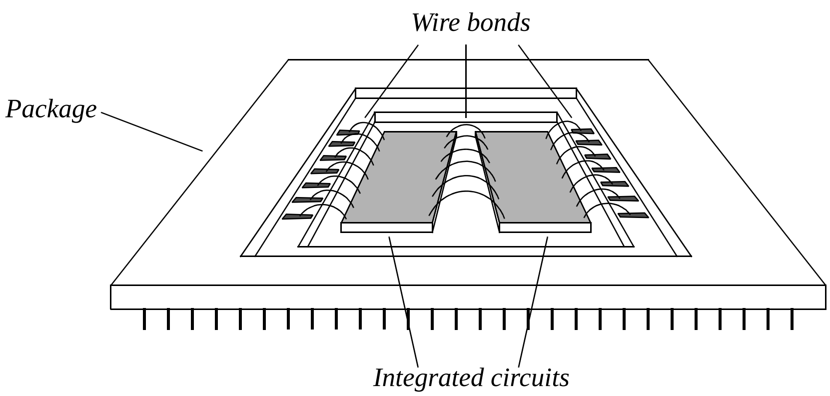


Figure 2-3. MCM with wire bonds

provides connections from chip to chip much in the same way as connections from gate to gate are made within the same IC, in the case of an MCM-D, as shown in Figure 2-4 [4].

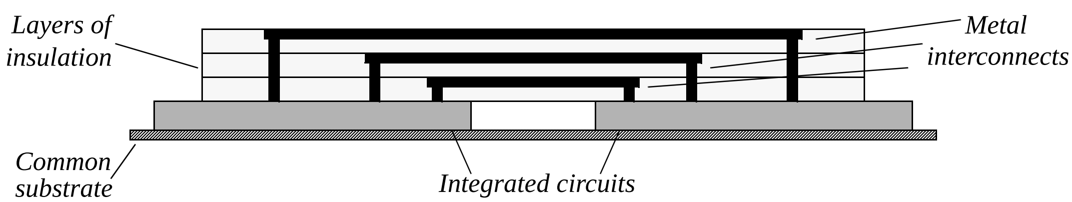


Figure 2-4. Cross-section of an MCM-D

The various MCM types offer a range of maximum operating frequencies and interconnect densities that are directly proportional to their cost -- the faster and more dense the interconnect is, the more expensive it is. As a result, low-end (wire bond) MCMs are considered to be a cost-effective improvement that offers performance advantages over the standard PC board implementation for circuits that require high bandwidth, such as a microprocessor to cache memory. A prime example of this type of MCM is Intel's Pentium-Pro, as shown in Figure 2-5 [5]. High-end (MCM-D) MCMs are not as commercially pop-

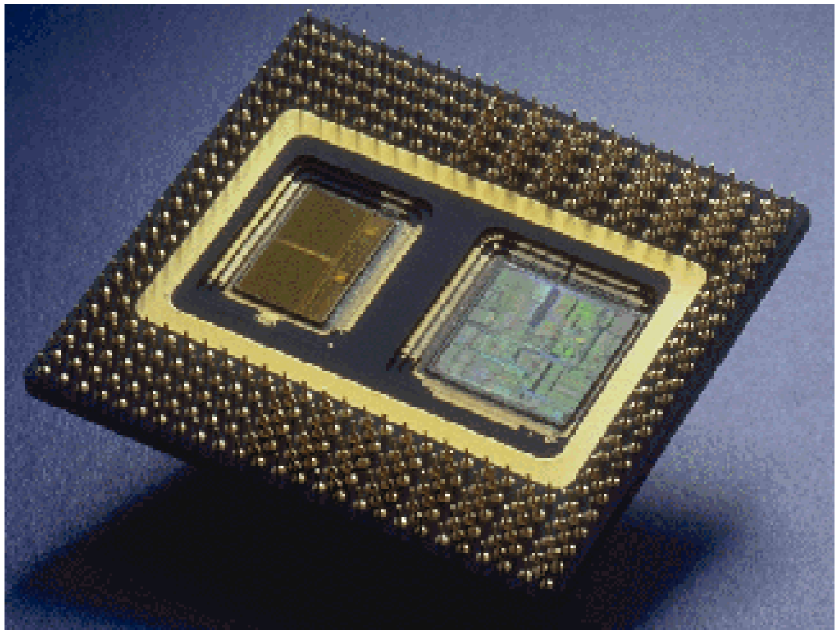

Figure 2-5. Intel Pentium Pro

ular although they have been developed in research.

### 2.2.2 Free-space optical interconnects

Another IC communication system that offers great promise is free-space optical interconnects (FSOI). A large amount of research has been done in this area and a lot of documentation has been written [6, 7, 8]. A summary is probably in order, though, because it will help the reader to understand the motivation and provide some context for optoelectronic VLSI. FSOI involves attaching (through flip-chip solder bonding) optoelectronic devices to metal I/O ports distributed in a gridded, area array on the surface of the IC. Figure 2-6 shows a 2k-bit first in, first out (FIFO) memory circuit with 128 area-distributed I/O pads. The attached devices either receive laser light or transmit it. FSOI circuits can also

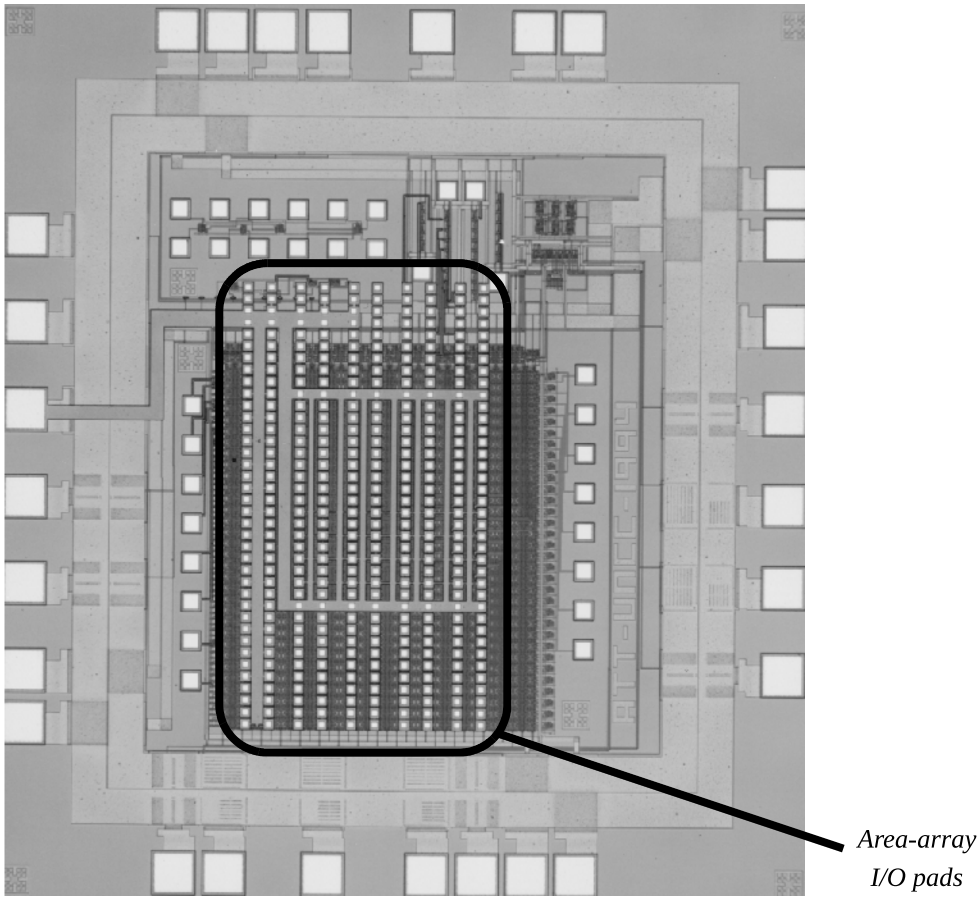


Figure 2-6. Microphotograph of an IC with area-distributed I/O pads

be categorized as “smart pixel” chips. Pixel is derived from the words “picture element”, meaning a point of light such as from a TV picture, and referring to the light that is transmitted off the surface of the chip. Smart refers to the fact that these pixels can drive some intelligent logic circuitry from signal processors to data routers.

One type of device that can either detect laser light or modulate it is the multiple quantum well (MQW) diode, shown in Figure 2-7 [9]. The MQW diode is a gallium-arsenide (GaAs) *p*-*i*-*n* diode, consisting of *p*-doped material, *n*-doped material, and *i*ntrinsic undoped material. When configured as a detector, the MQW diode converts optical energy

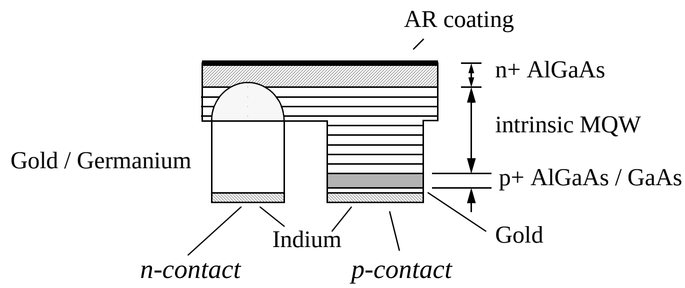


Figure 2-7. A multiple quantum well diode [9]

to electrical energy and generates an electrical current whenever light of a certain wavelength strikes the active area of the diode surface. That current gets converted to a CMOS voltage level through a transimpedance amplifier circuit called a receiver. The output of the receiver is then connected to the logic circuitry. In modulator mode, either optical energy is converted to electrical energy and most of the light is absorbed or the device is turned off and most of the light is reflected. The modulator requires an external laser to provide the light that is reflected and transmitted to the next IC. The modulator is driven by an electrical buffering circuit called a transmitter, which receives its input from the logic circuitry. A cross-section of an MQW diode attached to area pads is shown in Figure 2-8 [10]. The devices are solder-bonded to metal pads on the top layer of metal in the

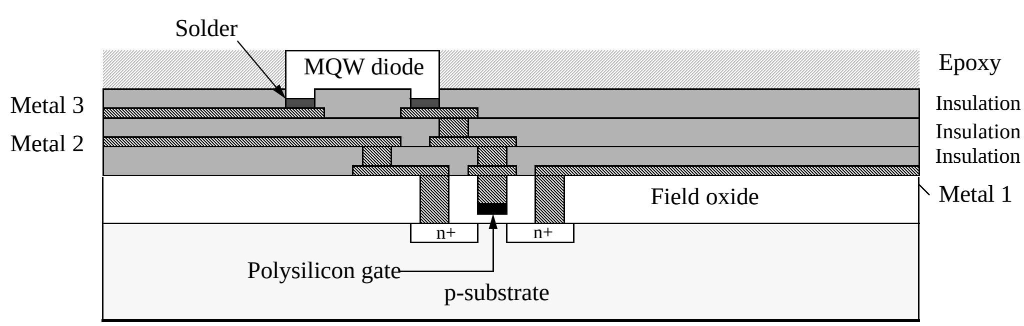


Figure 2-8. Cross-section of an MQW attached to a silicon die [10]

CMOS fabrication process. In this figure, the top layer of metal is metal3. Epoxy is added, as shown in Figure 2-8, around the devices as a protectant and sealant. Figure 2-9 shows

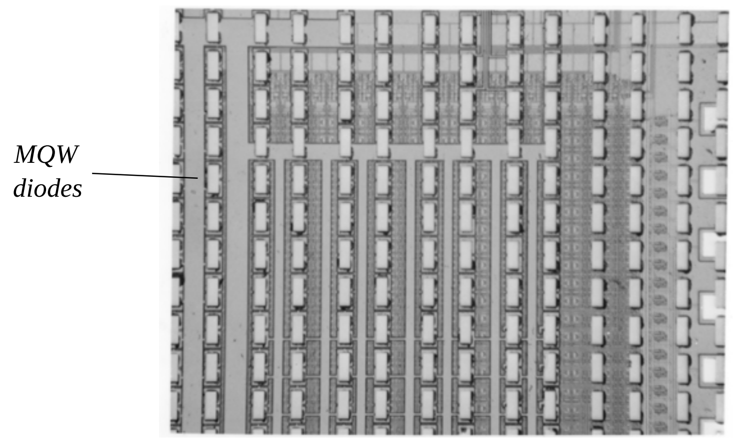


Figure 2-9. A microphotograph of an IC with MQW diodes attached to the surface.

the same FIFO circuit as in Figure 2-6 with MQW diodes attached to the area I/O pads. There are 64 diodes, 32 configured as detectors and 32 configured as modulators, attached to the 128 area I/O pads.

Another type of device that can transmit laser light is the vertical-cavity surface-emitting laser (VCSEL, pronounced “vixel”) [11,12,13,14]. The VCSEL is a *p-i-n* diode with mirrors, as shown in Figure 2-10.VCSELs have been flip-chip solder attached to the surface of a GaAs metal-semiconductor field-effect transistor (MESFET) circuit. However, there are problems in flip-chip bonding VCSELs to the surface of silicon CMOS, which is a far more desirable substrate to bond to because the overwhelming majority of ICs produced in the electronics industry are silicon CMOS. As a result, systems that incorporate VCSELs have been demonstrated using either wire bonding from a driver chip to a VCSEL chip or packaging the VCSEL in its own carrier and placing it on a PC board and connecting it as described in Section 2.1.1. Since the VCSEL is a laser, it needs no exter-

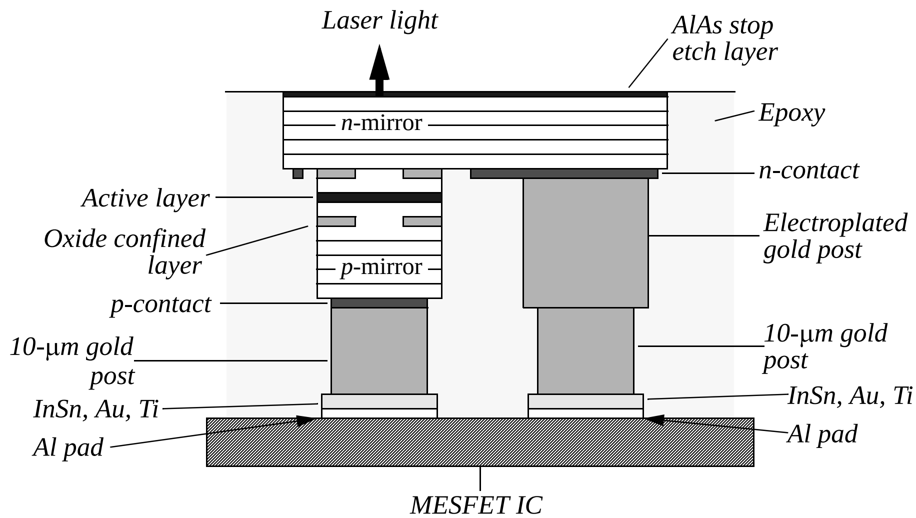


Figure 2-10. Vertical-cavity surface-emitting laser [11]

nal laser source and generates the laser beam itself. In this regard, the VCSEL has an advantage over the MQW modulator, which needs an external laser source. However, the VCSEL cannot be used to detect laser light as the MQW diode can. Furthermore, VCSELs consume more surface area than does the MQW diode and require more power to produce a laser signal. Overall, the VCSEL technology needs to mature somewhat before it becomes truly competitive with the MQW diode in terms of reducing system complexity by having the VCSEL bonded directly over silicon CMOS circuitry.

In an FSOI system, laser light that is transmitted from one IC is usually focussed through a lens above the surface of the IC and reflected by a mirror, or series of mirrors and optics, through another lens and onto the surface of another IC, as shown in Figure 2-11. The *optical components* in this figure could be beam splitters or halograms. The IC, with optical devices attached, is still perimeter wire bonded and packaged as discussed in Section 2.1.2. And the packaged ICs are still placed on a PC board. This allows some non-speed-critical signals and power to be supplied to the circuits through the wire bonds. All

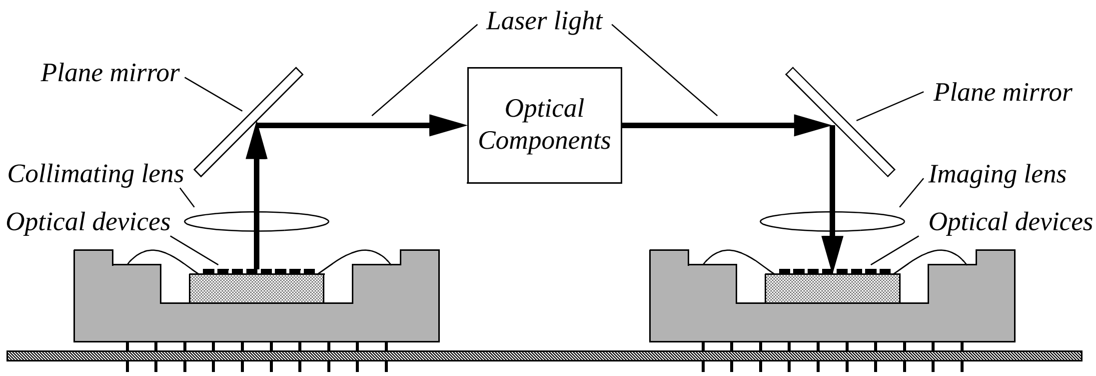


Figure 2-11. A free-space optical interconnect system[15]

the data signals are sent and received through the optical devices, lenses, and other optical components.

FSOI eliminates the communication bottleneck between chips on a PC board in two ways. First, it distributes the I/Os across the surface of the IC at a much smaller pitch than the pitch of I/Os bonded around the perimeter of the IC. The center-to-center spacing of perimeter wire bond pads is 100μm - 150μm [16]. MQW diodes can be spaced around 80μm apart [2]. Even though the optical devices are not placed over the entire surface of the chip, since they are on a finer pitch and placed in a two-dimensional array, area-distributed I/O can outnumber perimeter I/O on the order of thousands [2] per IC instead of just hundreds.

Secondly, each I/O is a high-speed connection that generally can switch at frequencies greater than that of the underlying silicon CMOS circuitry. In the Hewlett-Packard HP26 0.8μm process, the (single-beam) receiver circuits have been measured to operate at 550MHz [17]. The MQW diodes can switch at greater than 1GHz. Silicon CMOS ICs are generally operating at a maximum of 330MHz, in a much smaller process [18]. So as the results of developing the optical circuits in smaller processes (e.g. 0.5μm and 0.35μm) are publicized, one should expect the operation of these circuits to increase into the gigahertz range. Thus, there will be no bottleneck for the data entering and leaving the IC since the data can travel through the I/Os at a much greater rate than the logic circuitry can process it.

Finally, there is one more reason that the FSOI structure is such a powerful one -- even more so than MCMs. Low-cost packaging for coupling MQW modulators directly to single-mode fibers has recently been demonstrated [19]. In this fashion, there can not only be high-speed communication between chips in close proximity, such as on a PC board, but there can also be a fast link between chips over great distances. The implications of this long distance high-speed link are enormous for data networking, hopefully improving the performance of local area networks (LANs) from tens of megabits transmitted per second to hundreds and maybe thousands of megabits transmitted per second. By extension, chip-to-fiber coupling should enhance the bandwidth of large networks such as the Internet, by increasing the throughput of data routers and switches.

# CHAPTER 3 - DESIGN PARTITIONING

The VLSI for optoelectronic design methodology consists of three phases: partitioning, physical design preparation, and physical implementation. The partitioning and preparation phases can be performed simultaneously, but they have to be performed before the physical implementation can occur.

The optoelectronic custom IC design procedure has to begin at the highest level. Given a set of specifications for the circuit to follow, the design must then be partitioned into functional blocks. The way that the design is partitioned depends on the type of implementation at the physical level. Ordinarily this is not the case. Normally, the design should be partitioned into high-level blocks based on what logical function the block will perform and not on which transistor layout approach will be utilized or even what gates will compose the circuit. However, with optoelectronic VLSI, due to the various optical device connection strategies, the low-level physical implementation has a direct effect on partitioning the design at the high level.

The steps to follow for the partitioning phase of the design flow are listed in flowchart form in Figure 3-1, although not all steps are needed for every design. These steps are explained in detail in the following sections of this chapter.

## 3.1 Determining the smart pixel type

The choice of the smart-pixel layout approach will have an effect on the way the design is partitioned. And, the manner in which the design is partitioned will impact the lay-

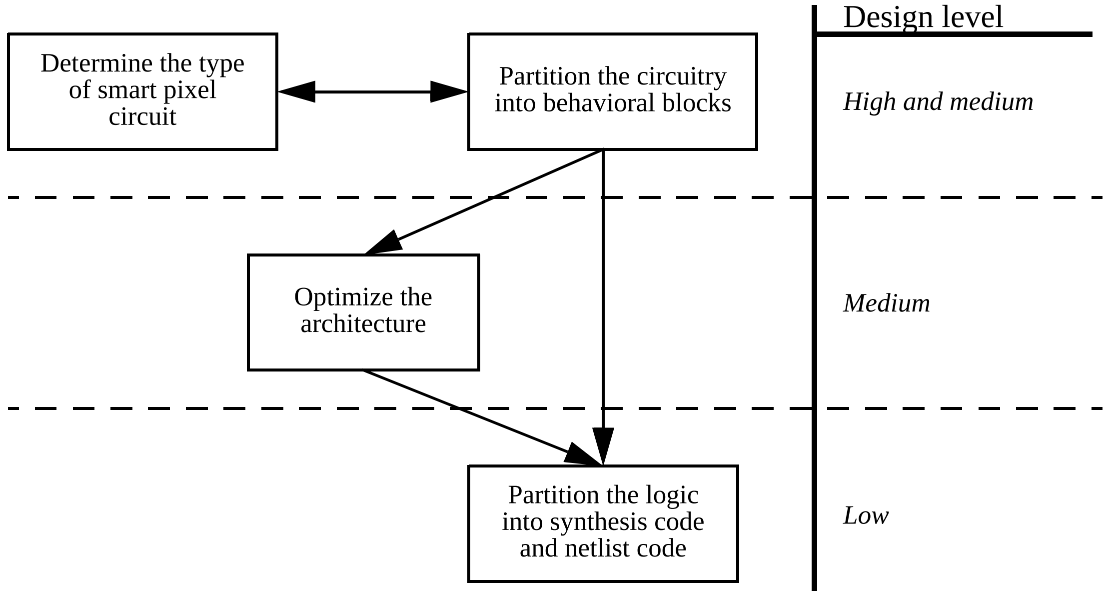


Figure 3-1. Partitioning flowchart

out. Therefore, the choice of smart pixel and the partitioning should be done in concert with each other. There are three different types of smart pixel architectures that have been explored through more than four years of research [20, 21]. The smart pixel is an FSOI circuit that is discussed in Section 2.2.2. Table 3-1 lists several ICs that have been de-

| | Functionality | Transistor Count | CMOS Feature Size | # of Optical I/O | Clock Speed | Off-chip Optical I/O Bandwidth |
|---|---|---|---|---|---|---|
| Processor | Processor Core [22] | 11k | 0.8µm | 192 | 100MHz | 19Gb/s |
| | DSP Core I [23] | 10k | 0.5µm | 78 | 100MHz | 8Gb/s |
| Memory | 2kbit FIFO [24] | 28k | 0.8µm | 64 | 50MHz | 3Gb/s |
| | 1kbit SRAM [25] | 10k | 0.8µm | 196 | 100MHz | 20Gb/s |
| | 2kbit Cache | 12k | 0.8µm | 192 | 100MHz | 19Gb/s |
| | 128-bit Register File [26] | 14k | 0.8µm | 64 | 277MHz | 18Gb/s |
| | 16kbit SRAM [20] | 180k | 0.8µm | 1,008 | 100MHz | 50Gb/s |
| Switching | Self-Routing Crossbar I [27] | 80k | 0.8µm | 340 | 50MHz | 17Gb/s |
| | Self-Routing Crossbar II [28] | 120k | 0.8µm | 1,024 | 50MHz | 34Gb/s |

Table 3-1: Summary of demonstrated ICs with area-distributed I/O pads

signed and tested at UNC Charlotte using manual layout approaches and fitting into various smart-pixel categories before developing the automated technique presented in this dissertation. Each of these smart-pixel architectures is suited for a specific application and involves distinct trade-offs. The paragraphs that follow review these layout methods.

The first layout scheme is to design a self-contained "pixel" circuit with electronic processing circuitry, input pad receiver circuitry for two or three inputs, and output pad driver for two or three outputs. This circuit is then replicated in a two-dimensional "pixel array" structure that matches the pitch of the area-distributed I/O padframe as shown in Figure 3-2. This approach was used to design the 277MHz 128-bit register file IC. The

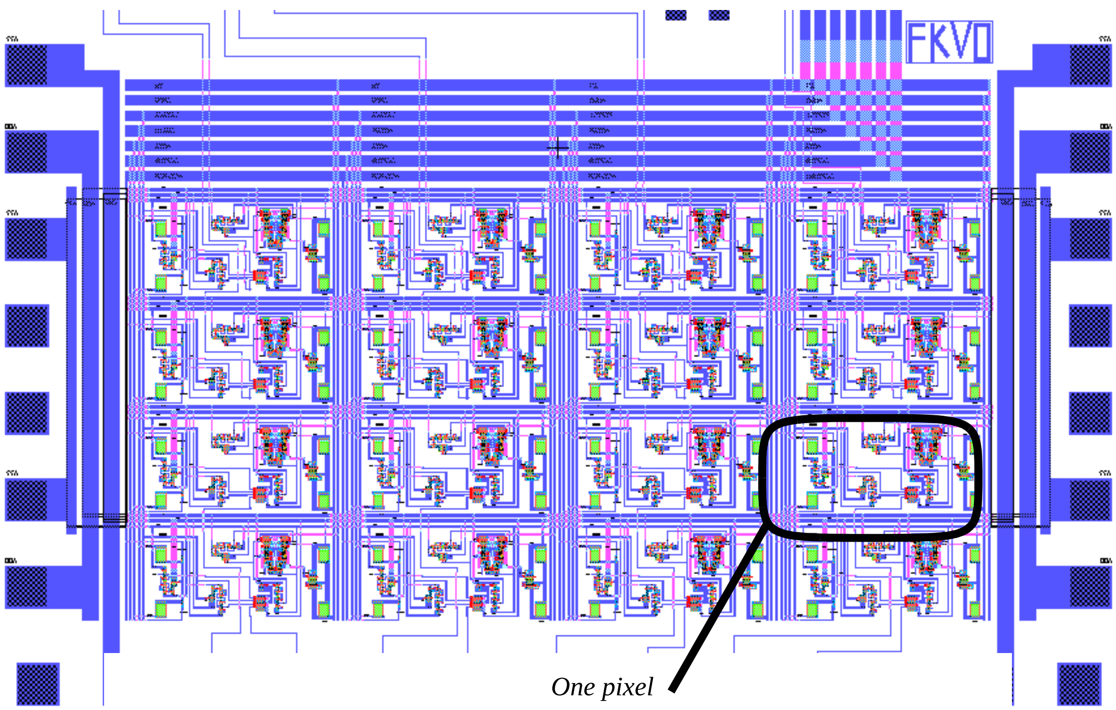


Figure 3-2. Layout of a pixel array

placement and routing of this IC required full-custom, manual layout effort. While this scheme is highly effective for bit-serial processor and register-based memory arrays, it is difficult to use it for the layout of large-scale VLSI circuits such as bit-parallel datapaths, crossbars, RAMs, megacells and cores. These VLSI circuits have custom layouts that have been developed to maximize their performance. Modifying these layouts to fit the “pixel array” layout would quickly become an impossible task. The challenge lies in dividing a large circuit into many small sub-circuits that are replicated in a two-dimensional array while still retaining efficient routing and performance characteristics.

The problem is made even more difficult because flip-chip I/O pad arrays typically use a fixed horizontal and vertical spacing (or pitch). Thus, in order to make efficient use of chip area, each “pixel” in the “pixel array” must contain roughly the same amount of logic and routing. It is difficult to meet this condition for large-scale VLSI circuits that use non-regular circuits and routing. To illustrate the above problem, consider the task of designing a 64-bit microprocessor core IC. In this design, most of the chip area is taken up by the 64-bit ALU and the 64-bit register file circuits. Typically, these VLSI circuits use a datapath layout style that creates a highly regular row and column structure. The datapath layout style is preferred for multiple-bit processing circuits because it achieves uniform timing for all bits in a word and because it minimizes routing congestion for these types of circuits. Partitioning this circuit to fit the “pixel array” layout scheme would result in non-uniform timing and increased routing congestion.

The second smart pixel implementation is to integrate the area-distributed padframe directly over active silicon VLSI circuits. This layout scheme was used to design the smaller ICs (less than 50K transistors) in Table 3-1. The placement and routing of these ICs was performed by Duet Technologies’ Epoch. However, routing to the area distributed I/O padframe positioned directly over the silicon circuitry required full-custom, manual layout effort. For example, Figure 3-3 shows a 1kbit static random-access memory

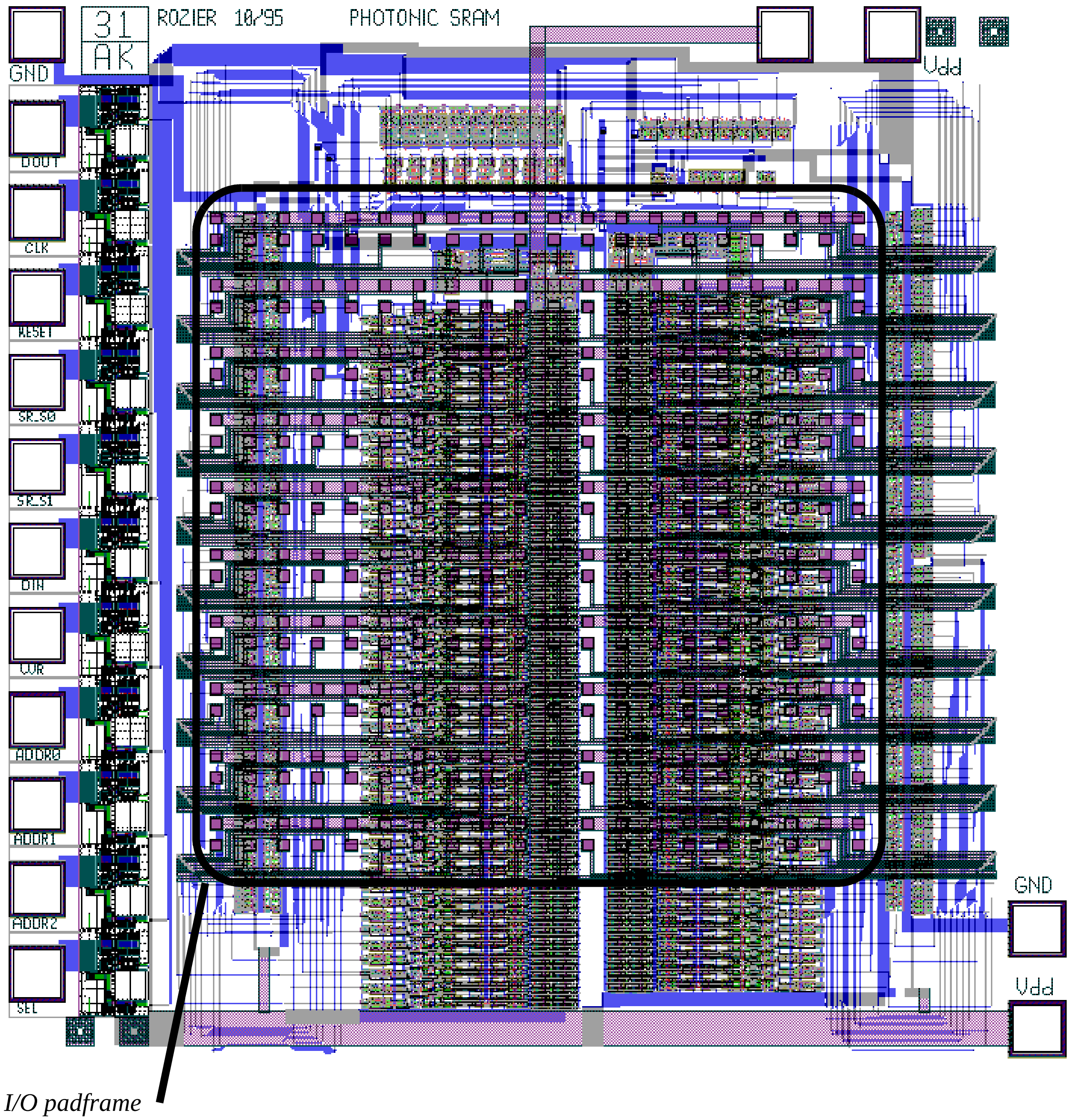


Figure 3-3. Layout of a 1-kbit SRAM

(SRAM) IC with 8 words and 96 bits/word. The 192 I/O pads on this IC are positioned directly over the SRAM circuit and operate at 20Gbps data rate. The ability to place I/O pads directly over active CMOS circuits provides the IC designer with considerable flexibility in the layout of the VLSI chip. This method enables the use of existing VLSI circuit layouts but it requires additional interconnect between I/O pad circuits (e.g. input receiver and output driver) and their corresponding I/O pads.

The third type of smart pixel is especially well suited for integrating distributed area I/O pads with high-density VLSI layout structures, such as bit-parallel datapaths, crossbars, RAMs, megacells and cores. This layout scheme was used to design the larger ICs (over 50K transistors) in Table 3-1. This method allows the I/O pad circuits (i.e. input receiver and output driver) and the core of the chip (i.e. logic and memory circuits) to be placed and optimized separately. It places the distributed-area I/O pads directly over their corresponding I/O pad circuits. The placement of the input I/O pads close to their receiver circuits and the output I/O pads close to their driver circuits improves signal integrity. The resulting two-dimensional array structure, called the pad interface module (PIM) is located in the center of the IC, or off to the side. The VLSI circuits are then placed around the PIM as shown in Figure 3-4, or they can be placed away from the PIM as shown in the layout of Figure 3-5. With this approach, the VLSI circuitry can be optimized independently of the pad interface module. Chip layouts generated using this methodology are area-efficient when distributed-area pad arrays with tight pitches are used.

One important advantage of the above method is its compatibility with current IC CAD tools. The placement and routing of ICs designed using this layout scheme can be fully automated. Although the design of the PIM still requires manual layout, once a standard PIM is developed it can be reused in different chip designs. For example, the 16kbit SRAM and the Self-Routing Crossbar I, listed in Table 3-1, use the same PIM. The PIM can be imported and treated as any other component by the CAD tool, that is described in Chapter 5.

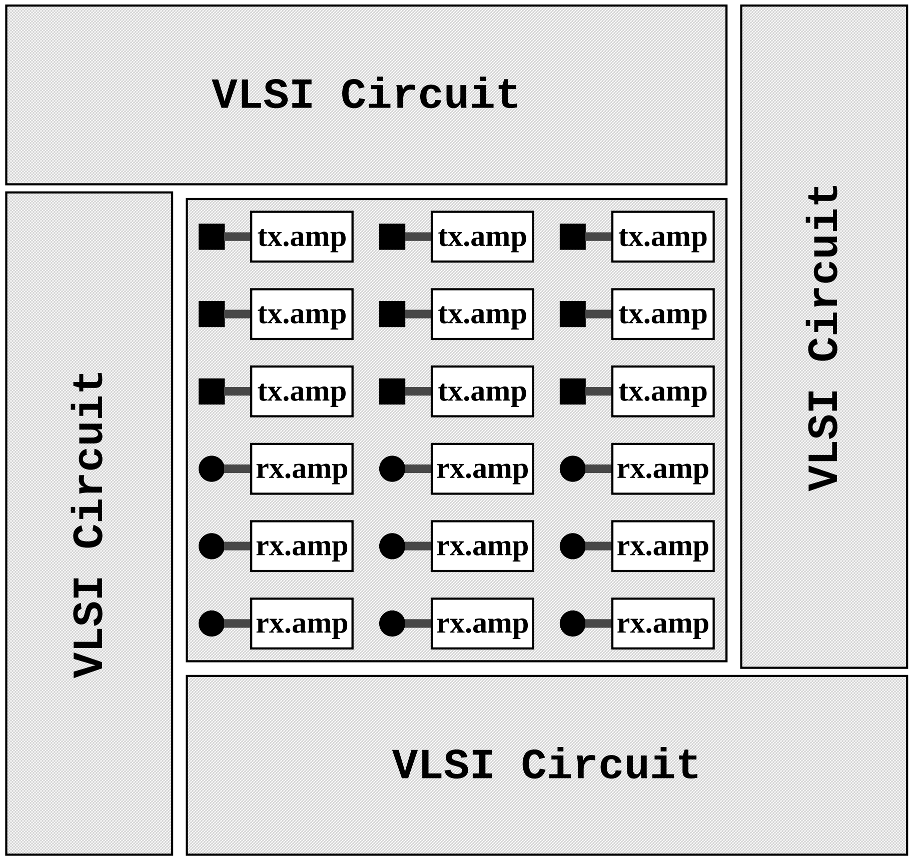


Figure 3-4. A smart-pixel IC model with a pad interface module

This layout methodology is especially well suited for the design of ICs that incorporate random access memory circuits, SRAM or dynamic RAM (DRAM). These circuits use weak internal analog signals that are amplified by sense amplifiers located at the periphery of the device. These analog signals would be corrupted by other active signals routed in their vicinity. For this reason, these circuits do not permit high-speed digital signals to be routed directly above them. The third type of smart pixel layout approach prevents this by ensuring that the placement of optical devices is restricted to the photonic interface module.

It is important to determine which type of smart pixel optoelectronic VLSI implemen-

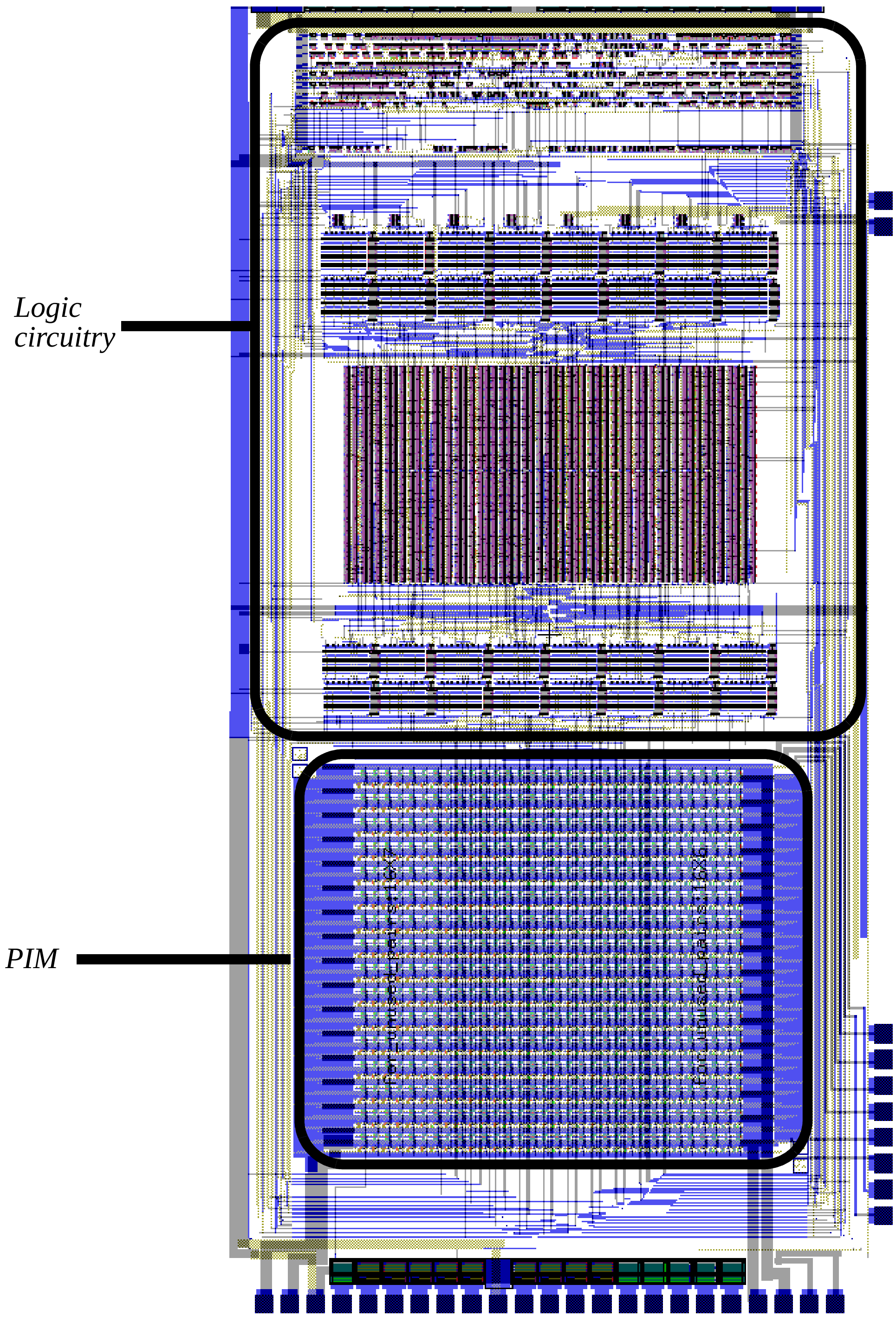


Figure 3-5. Layout of a self-routing crossbar with a PIM

tation is going to be used for the design in the early stages because this affects the types of high-level functional blocks that can be created, especially in the case of the pixel array.

The pixel array is best utilized for pure arithmetic or logic functions or temporary data storage, and due to the nature of the layout, there are only a few ways that the IC design can be partitioned. With the other types of smart pixel circuits, there are more choices for partitioning. There is more freedom in choosing the functional blocks and logic gates with the second and third types of smart pixels. However, the choice of smart pixel circuit has an affect on the rest of the design flow.

### 3.2 Functional block partitioning

This step is necessary for larger designs -- those that will have to be placed on multiple ICs, and as mentioned earlier, it will depend on the choice of smart pixel layout. It will probably be needed for most optoelectronic designs because the primary advantage of using optoelectronics for FSOI is that the connection from chip to chip can possibly be faster than the connection from gate to gate within the same chip. This step will not be needed for those smaller designs that will be contained on one chip.

One of the first things to consider when designing a circuit or system for FSOI is that the design will probably not be I/O limited. Since there are potentially thousands of I/Os per chip, the architecture of the system should effectively make use of all the signal lines that are available. This may mean allocating additional processing units to perform operations in parallel. It could also mean increasing the functionality of the system or enhancing the options that the system already possesses. Again, this affects the design flow at the upper levels.

The functional block partitioning involves determining which functional blocks will be placed on separate chips. With FSOI, which is really a specialized case of an MCM, comes a shift in the design paradigm that moves away from trying to place circuits that are tightly connected (i.e. there are a large number of connections or some timing critical connections between the circuits) on the same die. This paradigm shift has been studied in the case of MCMs [29,30,31]. A prime example of this shift is with the processor and memory system. In the past, it has been desirable to place the memory on the same IC as the processor to reduce the timing delay between these two units that communicate very heavily. The faster the processor could access data from the memory, the faster the system could operate. However, placing the processor and the memory on the same die has at least two drawbacks. One is that neither the processor nor the memory could be very large. There is a limit to the physical size that a die can achieve because the number of faults on a die increases with the area of the die. The second problem is that if the memory is going to be an SRAM type, then its performance won't be as great if it is fabricated in a logic fabrication process. Therefore, splitting the processor and the memory onto separate die will allow each one to be larger. The processor can have more registers or perform more arithmetic or logic functions, and the memory can have more capacity. In addition, the two can be fabricated in separate processes. It should be noted that SRAM built in an SRAM process is twice as dense and twice as fast as SRAM built in a logic process [31]. This block-to-die assignment is performed after the functional blocks have been decided upon, which will be in the later phases of the design process.

An example of applying this partitioning technique would be the design of a fast Fourier transform (FFT) processing system [32,33,34]. The Cooley-Tukey butterfly representation of a 16-point FFT calculation is shown in Figure 3-6 [32]. This algorithm converts time-domain data, *x(n),* into frequency-domain information, *X(k),* sixteen data points at a time. Each circle in the diagram is a radix-2 butterfly processor, shown in Figure 3-7

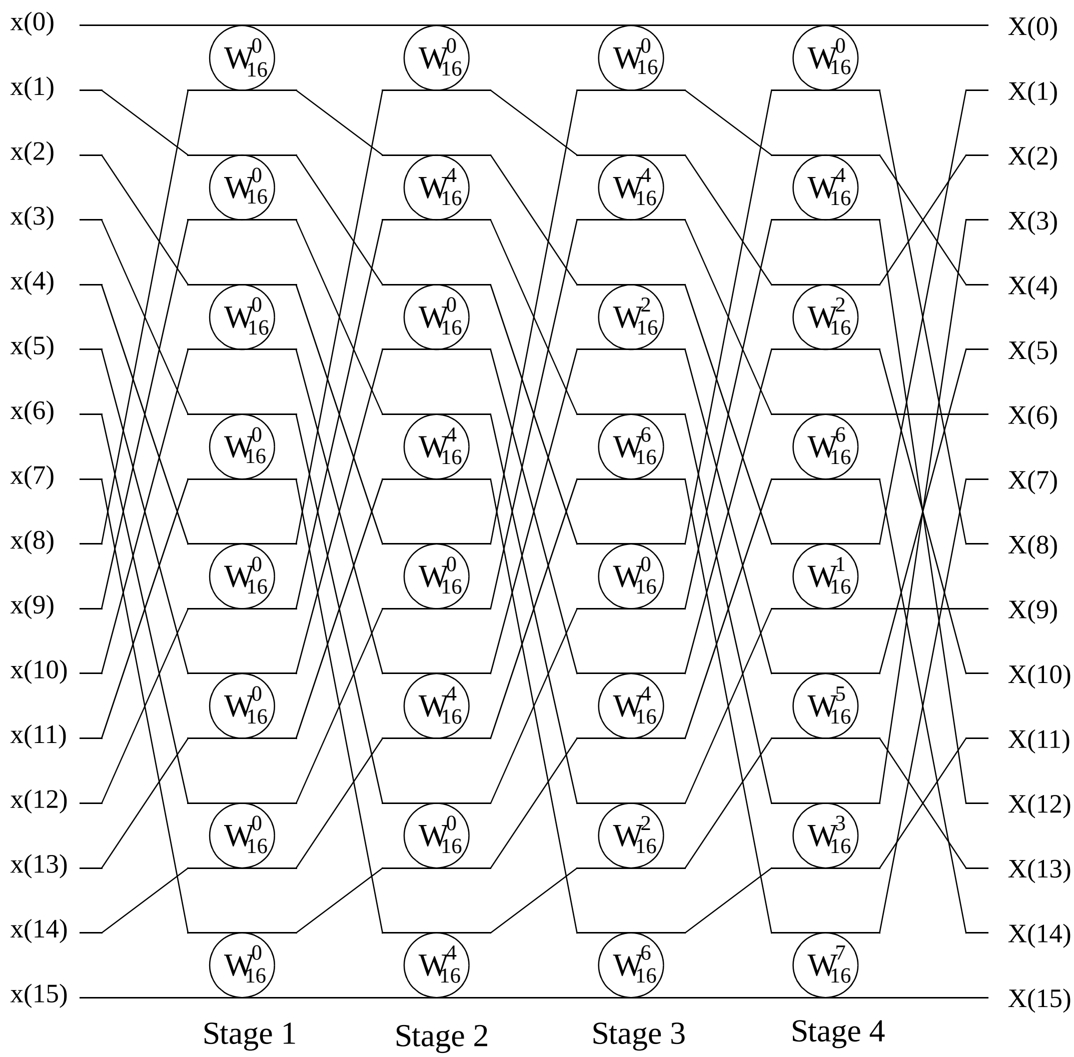


Figure 3-6. A 16-point FFT

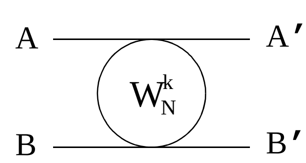


Figure 3-7. A radix-2 butterfly processor

which performs the following calculations from Equations 3-1 and 3-2:

$$A' = A + W_N^k B \quad \text{(EQ. 3-1)}$$

$$B' = A - W_N^k B \qquad \text{(EQ. 3-2)}$$

The FFT calculation is scalable in this fashion up to $N$ points, where $N$ is a power of 2. In this scheme, there are $N/2$ radix-2 butterfly processors per stage and $\log_2 N$ stages.

From Figure 3-7, it can be seen that the calculations at one stage of the FFT are dependent on the calculations from the previous stage. The intermediate results need to be stored between the stages. One way to partition the FFT logically is to perform all the calculations for one stage and store the results, followed by all the calculations for the next stage and storing the results until the last stage is done, as shown in Figure 3-8. The next

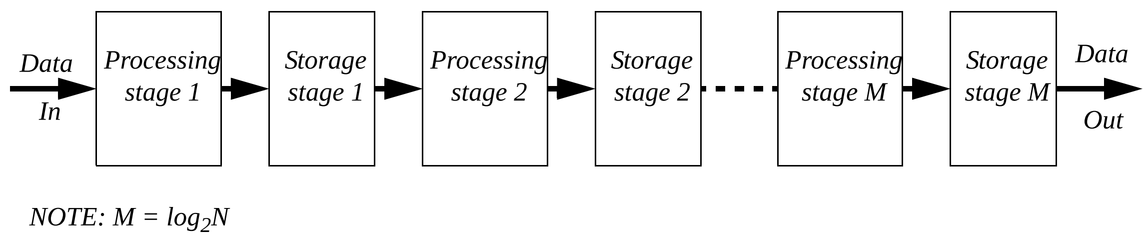


Figure 3-8. FFT logical partitioning

thing to do is functional block partitioning. There are a number of ways in which this can be done [34]. The entire calculation could be performed with one processor and two memory units, and they could fit on one chip. However, this would be slow and would not take advantage of the large number of I/Os that are available with FSOI. If the processor and the memory were on the same chip, the SRAM could not be optimized as mentioned earlier. So, splitting the FFT processor and memory into two separate chips not only allows the SRAM to be fabricated in an SRAM process but also means that the processor and memory can be made larger than if they were on the same chip. Therefore, more processors can be placed on the processor chip to perform up to 8 or 16 calculations in parallel since the entire stage of calculations of an FFT can be performed in parallel. This will greatly improve the throughput of the system. In addition, more memory can be added to the storage chip to accommodate all the data from a stage of computations. Finally, this partitioning choice allows the FFT processor to become a pipelined system to further increase data throughput as shown in Figure 3-9. Because it maintains the constant geometry

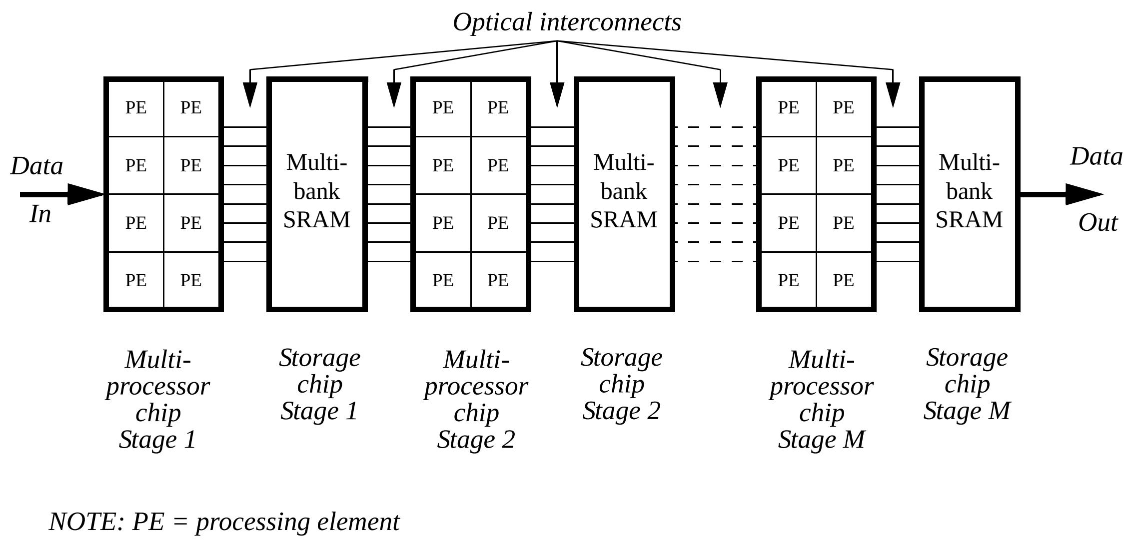


Figure 3-9. Functional block partitioning

of the algorithm of Figure 3-6 with no data delays or additional buffering, it is the most efficient implementation of the Cooley-Tukey FFT.

## 3.3 Architecture optimization

Once all the functional blocks have been assigned to chips, then the design within the chip must be optimized. For designs that are a sea of standard cells or mostly synthesized logic, the CAD tools can do the best job of optimization. For a pixel array type design, the only optimizing that needs to be done is in the choice of arithmetic or storage units. The designer can choose either smaller units that are slower or faster units which are larger, depending on the constraints of the design.

For other signal or data processing designs, there are greater choices in the logic implementation of the circuitry, which are based on the architecture. Trade-offs between the complexity of the circuit and its operating frequency have to be considered. Usually the more complex a circuit is, the slower it will run. For example, in the design of an FFT processor, such as the one discussed in Section 3.2, once the design was partitioned into functional blocks, like multiple processors on one chip and SRAM storage on the other chip, then the architecture of the processors had to be determined. There were choices to make about the radix number of the processors [34]. The radix number is the number of inputs and outputs that a processor has, as well as the number of calculations it does. A radix-2 processor has two inputs and two outputs, and performs two calculations. As the radix number increases, so does the processor's computational efficiency. The computational efficiency is a measure of the number of data points produced per arithmetic operation. The higher the number the more efficient the circuit is. In this case, it is most appropriate to compare the number of multipliers a circuit makes use of, because in hardware, the multiplier is the most space-consuming arithmetic unit. On the average, a multiplier is up to twelve times larger than an adder of the same word size. Therefore, it would seem to be advantageous to minimize the number of multipliers in a design. The radix-4 processor produces four data points and requires twelve multiplications, for an efficiency of 0.33. The radix-2 processor needs sixteen multiplications to produce four data points, for an efficiency of 0.25. However, the radix-4 processor has a more complicated twiddle factor, $W_N^k$, supply pattern. To properly supply the twiddle factors to the radix-4 processor would have increased the complexity of the control logic, making it run slower and possibly lowering the system operating frequency. Furthermore, it would have modified the system-level architecture because the radix-4 processing structure has fewer stages of calculations than the radix-2. As a result, it was decided that eight radix-2 processors on the processor chip was the optimal solution.

For RAM storage circuits, there is one point to make about their optimization. In general, most RAM structures, either SRAM or DRAM, cannot be cycled through at the same rate as logic circuits. For example, if a processor needed 32 bits of data at the rate of 300MHz (one word per clock cycle), there is no way that a RAM could supply that data if each address were 32 bits. In other words, a RAM cannot cycle through its addresses, either randomly or consecutively, that quickly. In the RAMs that were included in some of the circuits from Table 3-1, a 100MHz clock could be applied to the RAMs, but reading and writing data required more than one clock cycle, thereby reducing the effective data throughput to 50MHz or less. And an optoelectronic system needs large amounts of data very quickly. So the solution to this problem is to wire multiple RAMs in parallel [34]. By tying the address lines together, the bus width is increased and a lot of data can be transferred to the RAM chip during one clock cycle. This way the RAM chip can keep up with the bandwidth demands of the processor.

### 3.4 Logic circuitry partitioning

The last step to perform in partitioning a design for optoelectronic VLSI is to separate the logic or blocks that will be synthesized from that which will be instanced from the library of cells and megacells, as shown in Figure 3-10. In general, the control logic is broken down into a finite-state machine definition and synthesized into a netlist either automatically, through the use of a hardware description language (HDL) and a synthesis CAD tool such as Synopsys, or perhaps by a schematic capture tool. The methodology in this dissertation makes use of Very high-speed integrated circuit HDL (VHDL) as the

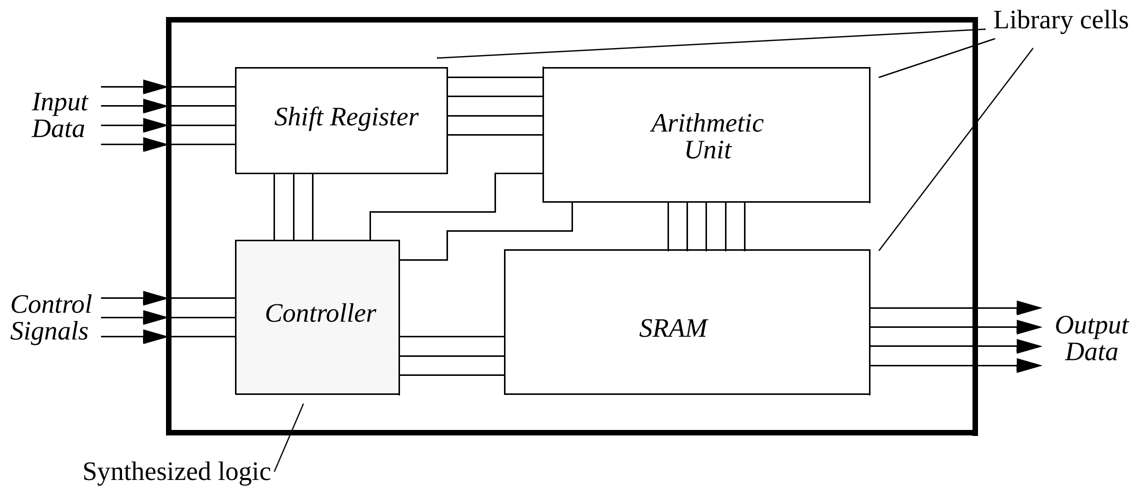


Figure 3-10. Block diagram showing library cells and synthesized logic

means to describe and define a design. It also uses Synopsys' *dc_shell* as a logic synthesis tool.

The rest of the circuit is composed of larger blocks such as shift registers, RAMs, or arithmetic/logic units. These blocks are already optimized for speed or area and do not need to be synthesized. They are entered into the design by selecting them from a library of parts and directly instancing them in the VHDL source code, as structural VHDL [35]. The library of parts is supplied by the transistor automatic place and route (APR) CAD tool vendor. The APR tool is the software program that performs transistor layout. Since it is not practical to build the circuit transistor by transistor, the APR tool needs a library of gates, cells, and megacells (that have already had transistor layout performed) to place and route. So the vendor creates a library for its APR tools to make use of. For this design methodology, Duet Technologies is the vendor and Epoch and Eggo are the CAD tools that are used.

# CHAPTER 4 - PHYSICAL DESIGN PREPARATION

Before the VLSI circuit transistor layout can be performed, the optoelectronic interface has to be prepared. The interface includes the area-distributed I/O padframe, the input conversion circuitry, and the output driving circuitry, as shown in Figure 4-1. Each of

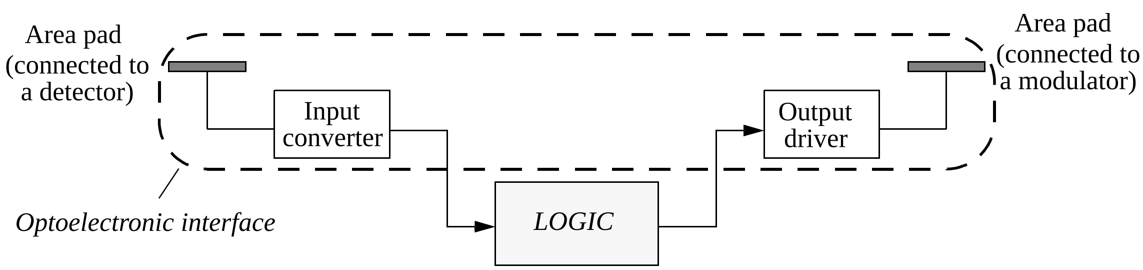


Figure 4-1. Block diagram showing the optoelectronic interface

these items needs to be created and imported into the Epoch environment database. However, once the interface circuitry has been created and imported, it can be used like any other Epoch library part, as discussed in Section 3.4, and does not need to be imported again unless there are changes to the layout such as reducing the feature size or modifying the layout to improve the performance of these circuits.

## 4.1 The Epoch project

Before proceeding with the discussion of the optoelectronic interface, the Epoch operating environment needs to be explained. While the design methodology presented in this dissertation is as automated as possible, it is not completely “hands off”. Some knowledge and manipulation of the database structure is necessary. Occasionally files in specific directories need to be edited. Epoch uses a *project* to keep its database in order. The project is one directory under which Epoch creates a number of subdirectories to keep track of the different blocks of the design. And a project must be created in order to use Epoch. The following steps cover how to create a project:

8. Choose a name for the project related to the design and create a directory. For example, *chipset* could be the name of the project.
9. Under the *chipset* directory, create another directory for the name of the ruleset, such as *hp14B.*
10. A directory needs to be created to store the VHDL source code. Under the *hp14B* directory, make a *vhdl* directory. At this point, the directory tree should look like what's shown in Figure 4-2.

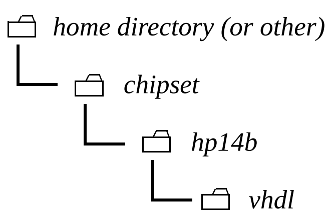


Figure 4-2. Preliminary directory tree

11. Run the Epoch graphical user interface (gui) by changing into the *vhdl* directory and typing: *epoch*. Select the *Project* pull-down menu and choose the *Project* Á option. Then click on *New...* In the *Project Path* field, type in the full path to the directory that will be the project. In this example, it would be: */afs/uncc/.../chipset/hp14b*. Note that although the Epoch interface is executed from the *vhdl* directory, it's not part of the project path. Now fill in the *Project Id* field with three characters of any sort -- letters or numbers. Click *OK*. To verify that the project creation was successful, check the *Project* field that is printed below the main black window. It should display the project

path that was just typed in. The pop-up window can be closed by pressing *Cancel*. Now the ruleset has to be chosen. Again, click the *Project* pull-down menu, and select *Ruleset...* From the list of rulesets available in the pop-up window, choose the desired ruleset and click *OK*. At this point Epoch can be closed if desired. There should be two results of all this effort. The first will be a file called *project.go* that is in the *vhdl* directory. The second is that there will be five new subdirectories created in the project directory. The directory structure should now look like Figure 4-3.

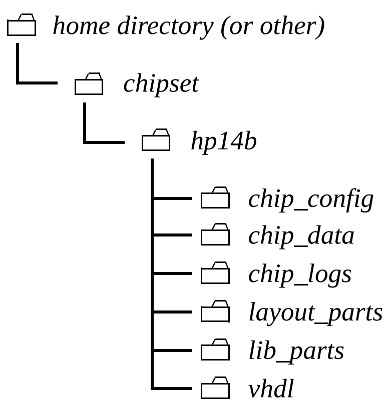


Figure 4-3. Directory tree after project creation

Once the project has been created, the Epoch executables will be run from the *project/vhdl* directory.

## 4.2 The area-distributed I/O padframe

### 4.2.1 The area pad

The optoelectronic padframe is a collection of pads that are distributed over the surface of the chip in a regular gridded fashion, although for special purposes they can be distributed in a non-regular or random pattern. Figure 4-4 shows just the area-distributed I/O padframe and the perimeter pads of the 1-kbit SRAM from Figure 3-2 (the rest of the layout is hidden). This is an example of a regular grid pattern. Each pad is a square of metal,

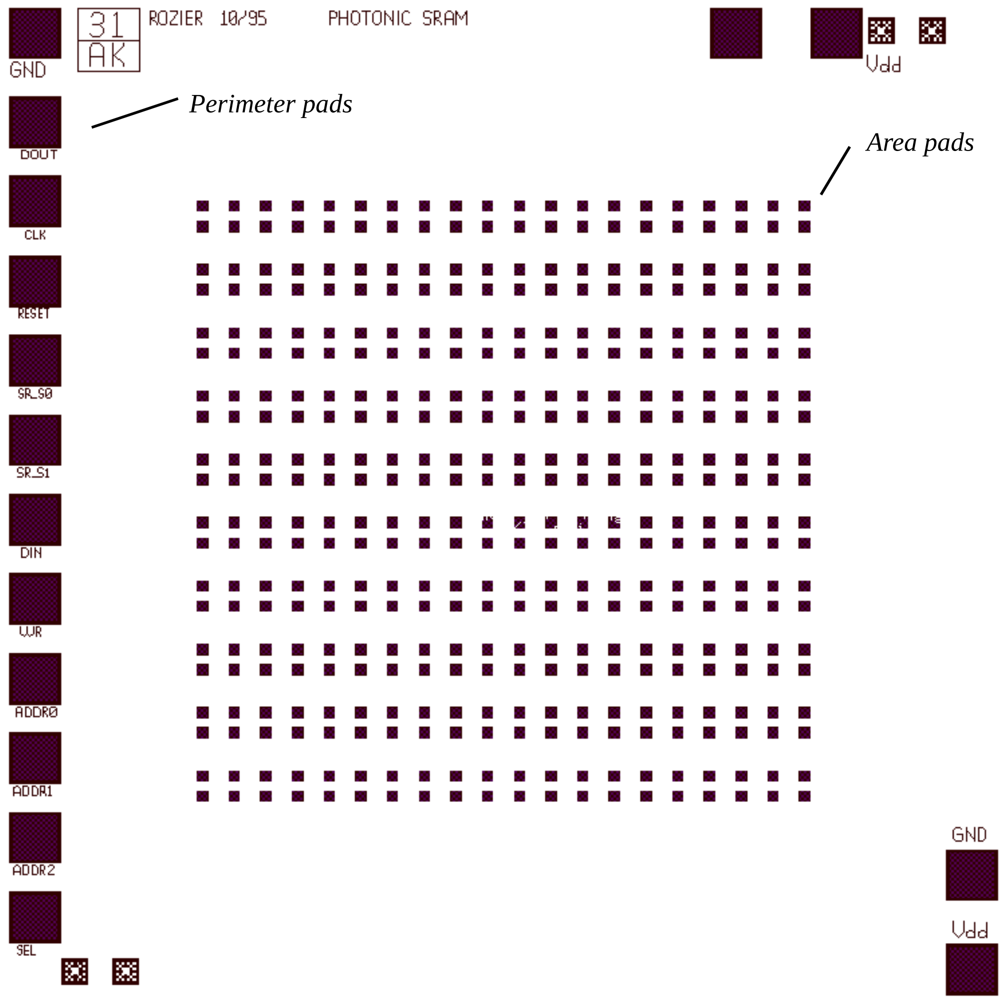


Figure 4-4. Layout of a 1-kbit SRAM showing the perimeter pads and the area pads (metal3 only)

always in the top layer of metal for the fabrication process. The area pad is created in a manual layout editor, such as L-Edit [36]. In the editor, all that needs to be done is to draw a square on the metal3 layer (the top layer for the Hewlett-Packard 0.8μm and 0.5μm processes), and add a glass cut on top of the metal that is a little bit smaller than the metal itself, as shown in Figure 4-5. Normally, the top layer of an IC is glass ($SiO_2$) insulation. However, in order to bond wires or devices either to the perimeter or the area pads, a re-

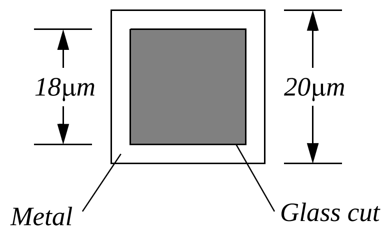


Figure 4-5. An area-distributed I/O pad

gion of the glass is etched away in to allow access to the metal conductor. The optoelectronic device attachment technology determines the size of the square. In most of the designs listed in Table 3-1, 20µm squares were used. Then, the file should be saved as GDS or GDSII (these two act interchangeably) data. When setting up the GDS output in L-Edit, click on the *Use default GDSII units (0.001*µ*)*. box. In order for the area pad to be properly imported into the Epoch database, the GDS layer numbers produced by L-Edit have to match the GDS numbers that Epoch is expecting. To make sure that there is a match, the numbers in L-Edit need to be checked against the *cmos.map* file. The *cmos.map* file is a text file that Epoch uses as a lookup table for translation between the Epoch geometry database and either CIF format or GDS/GDSII format. It can be found in the directory: $CASCADE_LIBRARIES/*technology*/*ruleset*, where *technology* is most likely *cmos*, and *ruleset* is whichever ruleset is going to be used, e.g. *hp14B*. If the numbers don’t match, then the numbers should be changed in L-Edit and the file should be saved again. Once the GDS data is correct, then the area pad is imported into Epoch as both an input pad and an output pad. The same piece of geometry can serve both functions. The physical aspect of the pad doesn’t change, it is still a square of metal to which an MQW detector or MQW modulator will attach. But, the logical function is different between the input and the output. The Epoch routing tool will wire the input pads only to the inputs of other gates or cells. The output pads can be driven only by the outputs of other gates or cells. The steps to follow to import a pad into the Epoch database are as follows (more complete directions on importing can be found in [37]):

1. Run the Epoch gui by typing *epoch* at the command prompt, in *vhdl* directory.
2. Click on *Input*, followed by *Cell Importer*... A large window will pop up.
3. In the box beside the text, *Name which imported cell will be called in Epoch*, type the component name that will be used in the source code. This name should probably reflect the function of the pad, such as *input_pad*.
4. Select #1 from the menu below the box (*Import GDSII and auto-generate import file*).
5. Click on *Link GDSII* File. This will launch a pop-up window. Point and click to select the GDSII data file. Press *Confirm*. This will create a new subdirectory in the project directory with the name that was typed in step #3. In that subdirectory, Epoch will put a link to the GDSII file.
6. In the box beside the text, *GDSII structure name to be imported*:, type the GDSII cell name.
7. Next to *Imported cell will be of type*: click *Pad*.
8. Click *Run Import*. If this finishes successfully, then proceed. Otherwise, correct any errors that may occur.

Once the preliminary importing is complete, a skeleton import file is created by Epoch and stored in the *project/cell_name* directory. It is called *ruleset*.*import*. In this example, it is called *hp14B.import*. The GDS/GDSII link that exists in the *project/cell_name* directory contains just the geometry of the cell. It says nothing about where to connect power and ground or any of the signals or even what type of cell this is. That's what the *.import* file is for. It provides descriptive attributes about the cell as well as the locations of all the ports. So it is necessary to edit the *.import* file and then update the database about the imported cell. After the *START_IMPORT* line, double check the following:

UNITS_PER_MICRON 1000

This line ensures that Epoch imports the dimensions of the object in the same units that L-Edit created it. The following attributes need to be added to the *hp14B.import* file, after the *START_IMPORT* statement:

ATTRCELL BOND_OPENING PADPIN

ATTRCELL PAD PAD

ATTRCELL PINNUM <integer>

These attributes tell Epoch that this is a bonding pad so that Epoch will allow this cell to exist on top of other cells -- normally something that would be avoided, because placing anything other than bonding pads over other cells would cause a collision of the geometry and the result could not be fabricated. Next, the ports need to be entered along with their locations. For bonding type pads such as these area pads, Epoch expects there to be a terminal named "PADPIN". This port determines the location of where the bonding wire will go, which doesn't matter in this case because no bond wire will be used on the area pads. So the PADPIN port is strictly for bookkeeping and has no significance otherwise. There also has to be a port that connects from the pad to the core circuitry. For those pads that are to be connected to output signals, then PADPIN needs to be listed as an output port. For those pads that are for input signals, then PADPIN needs to be an input port, as shown in Figure 4-6. The syntax for declaring a port is as follows:

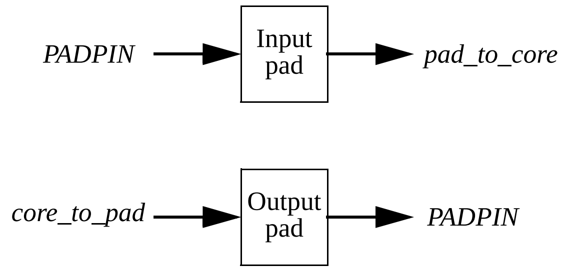


Figure 4-6. Area pads and signal directions

TERMINAL <routing_port> <electrical_node> <xloc> <yloc> <type> <layer1> <width1>

*TERMINAL* is a keyword identifying the start of a port declaration. The *routing_port* is the name assigned to the port itself. The *electrical_node* is the name of the signal that connects to that port. *xloc* and *yloc* determine the location of the center of the port in thousandths of microns. The *type* field indicates the kind of port that's being connected -- whether it's input or output, power, ground, or signal, and whether the port should be routed from the north, south, east, or west. The GDS layer number of the metal that will connect to the port is given by the *layer1* field. *width1* determines how wide in thousandths of microns the metal will be when it connects to the port. An example for the PADPIN port and the other pad port is shown below:

TERMINAL PADPIN PADPIN 10000 10000 inputports.nports.signalports 62,2000

TERMINAL padtocore_0 padtocore 10000 10000 outputports.sports.signalports 62,2000

Notice that for the PADPIN port only that the port name and the signal name are the same. Otherwise, the two names should be different. The rest of the information in the *.import* file is not as critical as what has just been presented and is adequately covered in the Epoch user's manual [38].

Now the database needs to updated with this new information:

1. Run the Epoch gui by typing *epoch* at the command prompt, in *vhdl* directory.
2. Click on *Input*, followed by *Cell Importer...*
3. In the column labelled, *Existing Imported Cells*, there should be listed the name of the cell whose *.import* file was just edited. Click on the cell name to highlight it.
4. Click on #6. *Update the database from an import file without rereading GDS-II and SPICE*.
5. Click *Run Import*. This should run without any major errors.

These steps should be followed for both the input and the output pads.

It is useful to have an idea of what the final area-distributed I/O padframe will look like in order to custom design the layout and improve the performance of the area-pad router, Eggo. The MQW diodes are two-terminal devices. One side connects to a power source and the other side either sends or receives the signal from a buffering circuit. Most likely, all the detectors will be connected to the same voltage source and all the modulators (or VCSELs) will be connected to a different voltage supply. Thus, there will be a lot of area pads that are electrically shorted together. This task can easily be accomplished with some clever creation of geometry. For example, with just the minimum of imported geometry, the layout would look something like Figure 4-7. In Figure 4-7, only the pads are placed and the router has to wire the power rail between the power pads. The dotted box to represent the attachment of the diodes is the same for all the pairs of power and signal pads but wasn’t drawn over the rest in order to cut down on the clutter of the figure. This is a less effective way to handle the routing of the area pads. For one reason, if the area-pad router has to route power lines as well as signal lines, it will take longer to run the program

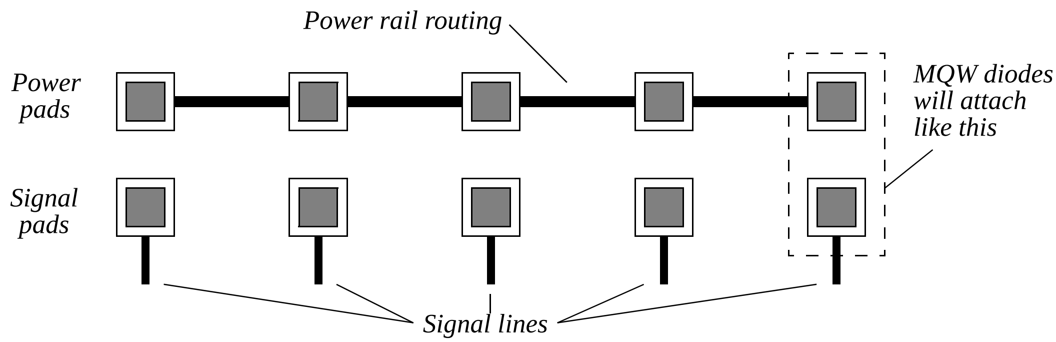


Figure 4-7. Layout of imported pads only

because routing the power lines doubles the number of nets that have to be routed. Secondly, it is more difficult to control the width of the power rails with the software, which is critical to supplying enough current to the MQW diodes. However, if the power rail is included with the area pad, then the width of the power rail can be controlled when the pad is drawn, as shown in Figure 4-8. The power rail shown in Figure 4-9 is merely a strip of

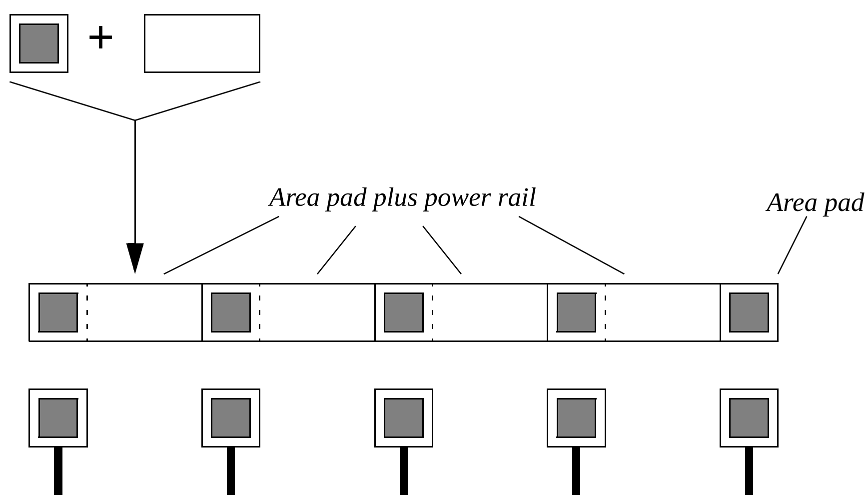


Figure 4-8. Layout of imported pads plus power rails

metal3. This also prevents the area-pad router from having to route the power rails, which will speed up the routing process. So two or more different types of area-pad cells are created for importing -- the area pad by itself, shown in Figure 4-5, which is used for signals, and the area pad plus a power rail wire. In addition, other shapes can be added that will help route the power rails to perimeter bonding pads or route the power rails to other power rails within the array, such as shown in Figure 4-9. Any shape of geometry can be

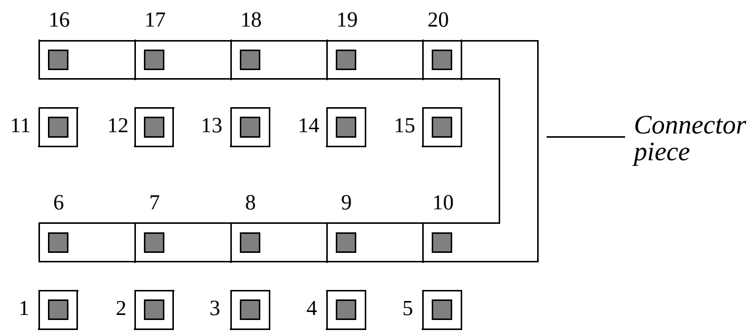


Figure 4-9. Layout with imported connector geometry

imported in the same fashion as described for the area pad and easily brought into the source code because each pad is a separate cell whose position is determined by its number attribute in the source code. So, in Figure 4-9, pads 1 - 5, 11 - 15, and 20 would be instanced as regular area pads. Pads 6 - 9 and 16 - 19 would be instanced as area pad plus power rail. And pad #10 would be instanced as area pad plus connector. More on the numbering of the pads is in Section 4.2.2. More information on instancing the pads in the source code is in Section 5.1.6.

It is best to lay down the power rails in this fashion before routing the signals because the signals will be routed mostly in metal3 and they will cross into the lanes where the power rails should be. Therefore, it won't be possible to place the power rails after the signals have been routed. But, the power rails have a higher routing priority than the signals because they are shorted together.

The power rail pads have to be handled specially. Since it is easier and more efficient to route power to the pads by importing pads with the power rail included, like in Figure 4-9, then the power rail pads will not have any signals wired to them by the area pad router. However, Epoch requires that the area pads have to possess an input and an output. So that eliminates the option of creating just a one-port pad. During the netlist check phase of the physical design process with Epoch, any components, gates, or cells whose output port is not driving another input are deleted from the design (unless the output of a gate is driving an entity-level port). If this happens, the design obviously won't work because part of it will be missing. There is no way to avoid using dummy signals and having them deleted. But, it's acceptable to Epoch if a cell's input signals are deleted. If the power pads are of the output type, then the PADPIN port will be connected to an output signal at the entity level. Therefore, the input signal to the pad will be deleted (which is acceptable), but the output signal won't be. So, the power pads have to be of type *output*, as shown in Figure 4-6.

### 4.2.2 The package file

The package file is what Epoch uses to determine the position of each numbered area pad. The file contains a list of numbers for pads (or pins as they are called in the file) and the matching *x* and *y* coordinates of the position of each pad. The physical spacing (and thus the locations) of the area pads are determined and given by the optical device fabrication engineers. These numbers are what is entered in to the package file. The *x* and *y* locations of each pad are given relative to the center of the design. Therefore, in order to properly position each pad, the designer should divide the *x* and *y* dimensions of the array by two and use the negative of those numbers as the lower-left corner. For example, in Figure 4-10, if the size of the array were 200 mils x 100 mils, the coordinates of the lower-

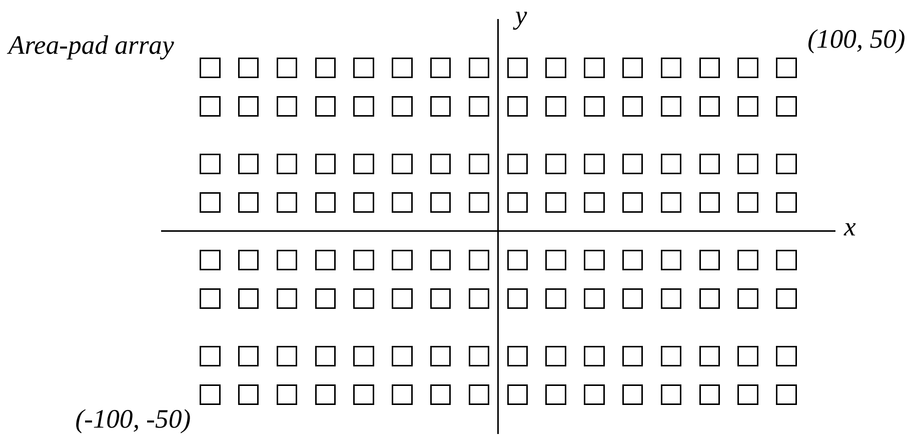


Figure 4-10. Coordinates of the area-pad array

left corner of the array should be (-100, -50) and the upper-right corner will be (100, 50). Then, all the rest of the pads should be placed with respect to the new lower-left corner.

Since the syntax for the area pad package file is the same as the perimeter pad package file, the user can retrieve a perimeter pad package file from the *$CASCADE/tech/packages* directory and use it as a template. The following is an example:

```
! PACKAGE_VERSION 1.0
PART_NO             none
MFGR                none
INT_PART_NO         uncc402
UNITS               mils
PACKAGE_TYPE        flip-chip
PACKAGE_LENGTH      12
PACKAGE_WIDTH       12
CAVITY_LENGTH       10
CAVITY_WIDTH        10
SUBSTRATE_PIN       0
```

| ! | pin# | side | xloc | yloc | height | width |
|---|---|---|---|---|---|---|
| PIN | 1 | A | -100.0 | -50.0 | 10 | 10 |
| PIN | 2 | A | -80.0 | -50.0 | 10 | 10 |
| PIN | 3 | A | -60.0 | -50.0 | 10 | 10 |
| PIN | 4 | A | -40.0 | -50.0 | 10 | 10 |

Comments are preceded by an exclamation point (!). A value in the *PART_NO* and *MFGR* fields is optional. The *INT_PART_NO* is the name of the package. The *PACKAGE_TYPE* for area pads is *flip-chip*. This file allows the user to change the units from mils to microns. However, this is not advisable since although Eggo can understand either unit, the executables in Epoch only work in mils. The *PACKAGE_LENGTH*, *PACKAGE_WIDTH*, *CAVITY_LENGTH*, *CAVITY_WIDTH*, and *SUBSTRATE_PIN* fields refer to perimeter pad packaging. The *side* field has to be *A* for area pad. The *height* and *width* values of each pad are not used by the area pad router, but are added for completeness.

When creating the package file, numbers have to be assigned to the pads such that each number corresponds to a pad in a certain location. It is important to make a note of the number of each pad in the grid. Because if a certain signal has to be wired to a certain location or certain pad, then the each pad has to be identified in the VHDL source code by a number. The link between the location of each pad and the signal that is connected to it is made in the source code. Therefore, the array of Figure 4-10 should have numbers assigned to it as shown in Figure 4-11.

The package file has to contain the numbers and locations of all the area pads. For any designs that contain more than 20 or 30 pads, this could be a tedious exercise. An area-pad generator program could be written in a language such as C or Perl to ease the typing bur-

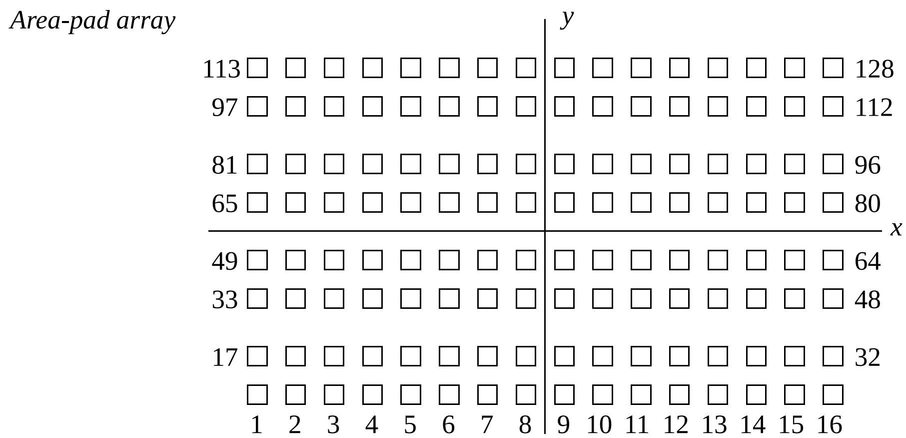


Figure 4-11. Numbering the pads

den. It should take in the total number of pads, the number of pads per row, the number of rows, the spacing between the pads, and the size of each pad. It should then produce the text that can go directly into the package file. An example of a generating program is in the appendix.

Once the editing of the package file is complete, then the file has to be given a name with a *.pkg* extension and stored in the *$CASCADE/tech/packages* directory.

## 4.3 Input conversion circuit

The laser detecting MQW diodes convert the optical energy (or photons) to electrical current (or electron flow). However, all of the digital logic circuitry operates from voltage levels. So a current flow into the input of a logic gate would not trigger a change in the output. Therefore, a circuit has to be added to the system to change the current flowing from the diode to a voltage that the logic inputs can use. That circuit is called a receiver. It is a transimpedance amplifier, which means it takes a current input and produces a voltage output, as shown in Figure 4-12. There have been numerous papers written on the various design aspects of receivers, such as power consumption, efficiency, and operating fre-

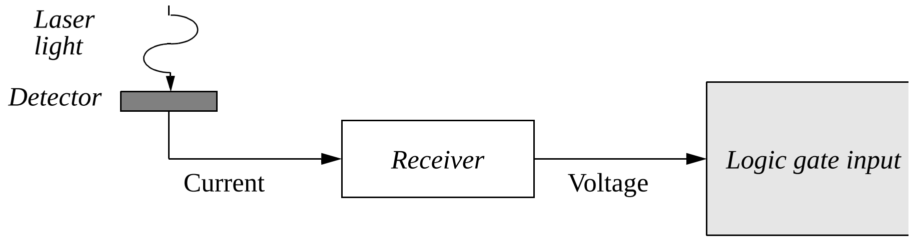


Figure 4-12. Receiver function

quency [1,17].

The receiver is an analog circuit that is created in a manual layout editor, such as L-Edit. It should also be simulated before it is imported into Epoch. The load at the output of the receiver can be simulated as a capacitance that represents the capacitance of the MOSFET gate of the logic input. Once created, it must be saved in GDS format as described in Section 4.2.1 for the area pad. Then, the same steps for starting the import should be followed as listed for the area pad in Section 4.2.1, except number 7, which should be: click on *Block*. Once the *.import* file is created, then it should be edited. The following two lines should already exist in the file:

UNITS_PER_MICRON 1000

ATTRCELL CELLTYPE BLOCK

If they don't appear at the top of the file, then add them after the *START_IMPORT* line. Next the locations of all the ports, including Vdd and GND, need to be added with *TERMINAL* statements. In addition, *NODE* statement should be added to give the capacitance and impedance values for the ports as described in the Epoch manual [39]. When the editing is complete, the database should be updated as described in Section 4.2.1.

4.4 The output driving circuit

The output driving circuit, also known as a transmitter, is simply a standard CMOS buffer, which generally is two inverters in series. Its function is shown in Figure 4-13. For

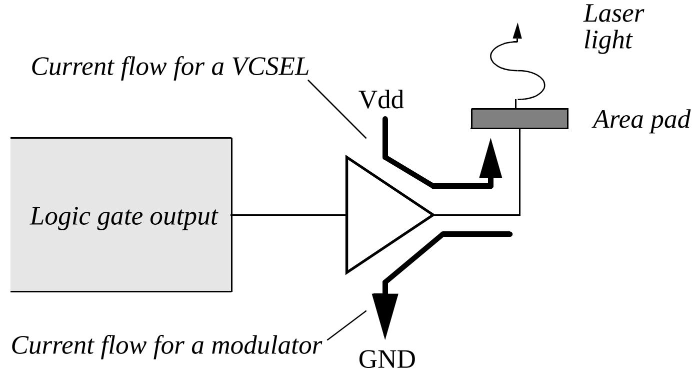


Figure 4-13. Block schematic for a transmitter

the MQW diodes, the buffer has to sink a lot of current when the diode is not modulating (reflecting). For a VCSEL, it has to supply a lot of current when the laser is on. Since the Epoch library has a large supply of buffers there is really no need to import a buffer. If the amount of current that has to be handled by the buffer is already known, then a size can be chosen by the designer in the source code. If the amount of current is not known and the amount of capacitance of the load is known, then that value can be entered into Epoch, and Epoch will automatically choose the appropriate buffer size. To have automatic buffer sizing performed, follow these steps (assuming the source code is already written and the design has been entered through the “netinput” stage):

1. Run the Epoch gui by typing *epoch* at the command prompt, in *vhdl* directory.
2. Click on *Physical Design*, followed by *Parameters* Á *Timing* Á *Define outputs*...
3. In the window that pops up, click on the name of the design in the *Existing Designs* area.
4. All of the output signals listed in the source code will appear in the *Primary Outputs* area. Select the name of the signal that will drive the modulator.

5. Type in a value for the *Maximum Delay Constraint*, that is about two times as much as the clock period.
6. In the *Output Load* field, type in the capacitance value of the modulator.
7. Click on *Add*.
8. Repeat this for all other signals that drive modulators.
9. Click on *Save*.

When running the automatic layout compile, select *Timing driven* buffer sizing. This will choose the size of the buffer based on the capacitance values.

# CHAPTER 5 - PHYSICAL DESIGN IMPLEMENTATION

The physical design implementation is the last phase of the design process. It involves the source code creation and the steps to follow to perform the transistor and area-pad layout, as well as a means for simulation.

## 5.1 VHDL source code

There are a few tasks that the source code has to perform. The first is to obviously describe the circuitry in either a behavioral or structural VHDL fashion. Also, it has to establish hierarchy by partitioning the design into multiple source code files. Another job that is done in the source code is to maintain the grouping of those gates that need to be placed together at the transistor level. Finally, for optoelectronics, it has to provide all the attributes for the area-pad package as well as the area pads themselves.

### 5.1.1 Establishing hierarchy

The source code will most likely be written in more than one file. Establishing the design hierarchy and partitioning the functional blocks occur at this point. The part of the design that needs to be synthesized, such as control logic or special mathematical functions that don't exist in the Epoch library, has to be written in its own behavioral VHDL source file, which is synthesized. The synthesized logic is then a component that is instanced in structural VHDL (in a separate file) in an entity that is at a higher level in the hierarchy. Other functional blocks are written in structural VHDL and combined in the source code in an upward fashion in the hierarchy. Figure 5-1 shows a Venn diagram of an example

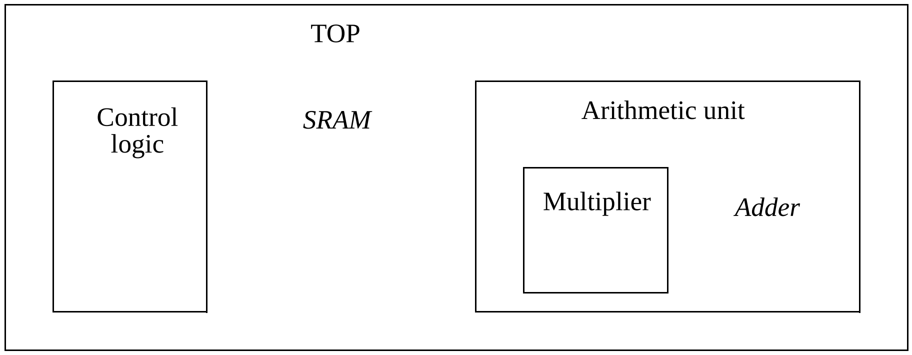


Figure 5-1. A design hierarchy

DSP-type design. The boxes represent separate files. TOP is the highest level of hierarchy. In the source code for the TOP entity, the controller is instanced as a user-created component, along with the arithmetic unit, as shown in Figure 5-2. The SRAM is instanced as an Epoch library component. The controller is a separate entity that is synthesized logic. The arithmetic unit has two components in it -- the multiplier which is synthesized and the adder which is an Epoch library part. The multiplier is also a separate entity. After synthesis, what is actually included in the design is the netlist that Synopsys creates. Epoch needs a structural netlist of gates to perform placement and routing because it can't understand behavioral VHDL. The netlist tells Epoch exactly what components to place and how to connect them.

### 5.1.2 Physical placement partitioning

While logic partitioning can be accomplished by creating separate source code files with separate entities, there is another reason to create separate entities. Physical partitioning can be achieved also. It is a good idea to keep those gates close together that have a high amount of connectivity or that have critical high-speed communication. Epoch, by default, mixes the standard cells from different components together when it performs lay-

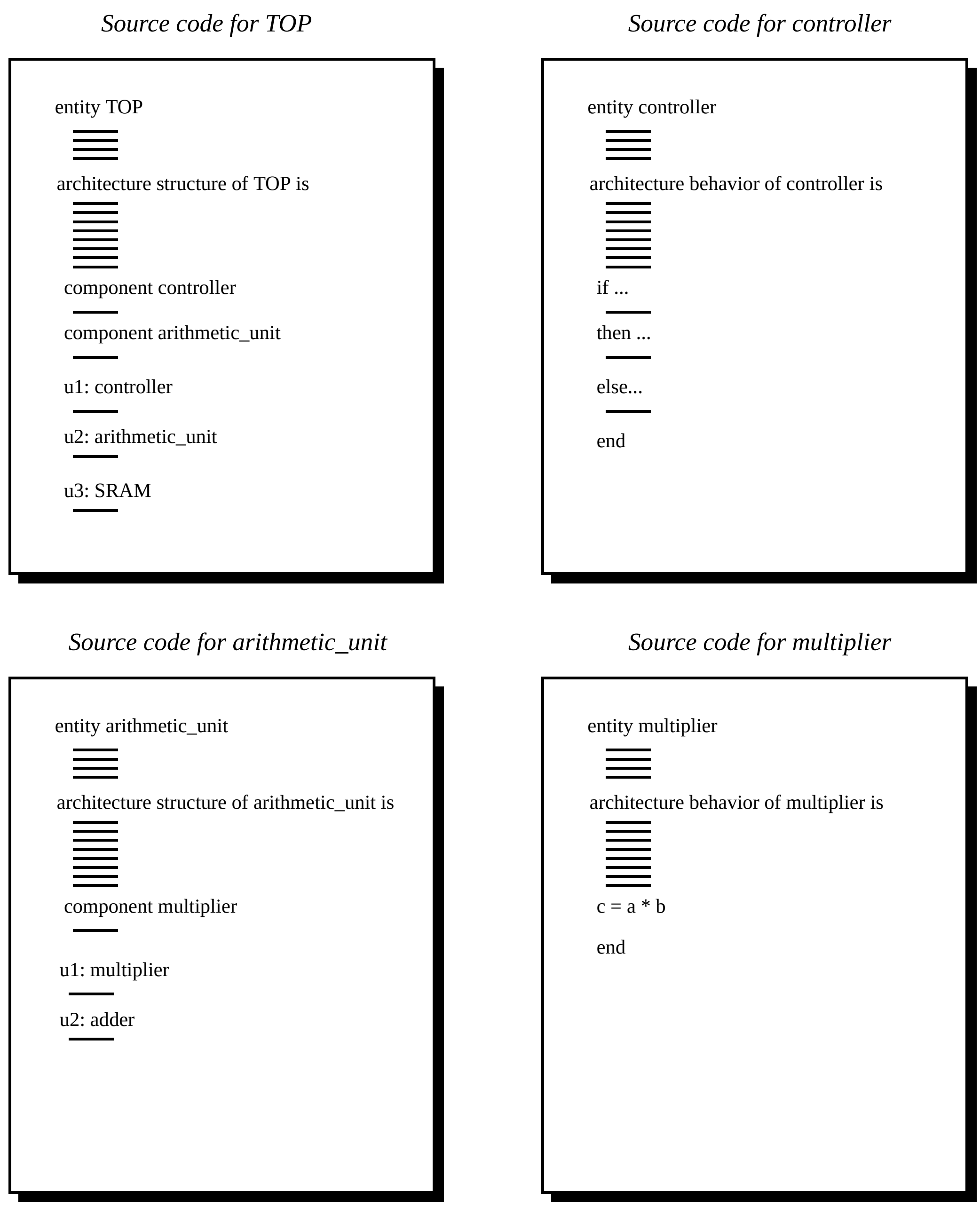


Figure 5-2. Sample VHDL source code

out. So that, in the example of Figure 5-1, the gates from the synthesized multiplier would be combined with the gates from the controller, which would probably be inefficient for both components. Placing all the standard cells together in one group would make the layout smaller. But, for high-performance designs, the lumped group placement is not desirable because it leads to longer propagation delays, which in turn leads to lower performance. In addition, Epoch tries to optimize for area when floorplanning the entire circuit. In multi-component designs such as the one in this example, each lower-level entity is compiled and floorplanned before the upper-level entity that instances it. This has to be done for all lower-level components and allows for optimization for speed at each level of the hierarchy. However, for the entity *TOP*, Epoch will randomly place all of the components, whether they be standard cell groups or larger blocks, such that the area consumed by the circuit is the smallest. This might mean that the adder and the multiplier in the arithmetic unit are placed far away from each other in order to minimize the circuit area. Again, this could lead to lower performance because of large interconnect delays. The way to prevent Epoch from lumping all the standard cells together as well as floorplanning for smallest area is to add *fixed block* and *cda_module* attributes to the source code. The *fixed block* attribute is added to the entity that has to remain intact. In the above example, the *fixed block* attribute would be added to the VHDL netlist file for the multiplier, the VHDL netlist file for the controller, and the source code for the arithmetic unit. The attribute is inserted in the entity declaration, as shown for the arithmetic unit:

```
entity arithmetic_unit
    port( ...
    );
    attribute fixedblock : integer;
    attribute fixedblock of arithmetic_unit : entity is 1;
end arithmetic_unit;
```

The *fixed block* attribute has to be declared as an integer first. The integer value of 1 means that *fixed block* is assigned to *arithmetic_unit*. If the value were 0, the attribute would not be applied. Now, when instancing the component *arithmetic_unit* in the architecture of another entity, the designer has the option of either maintaining the integrity of the component's original placement by using the *cda_module* attribute or having the component mixed in with the rest of the circuit by not using the *cda_module* attribute. This allows flexibility in the design process because the same component might be used in many different designs. In some cases, it might be best to preserve the original layout and in other cases, it might not. To keep Epoch from destroying the original placement of a component, the *cda_module* attribute is added in the architecture of the upper-level entity before the *begin* statement. In this example, the attribute would be included in the architecture of the entity *TOP*:

```
architecture structure of TOP is
    component arithmetic_unit
    port(
    );
    end component;
    attribute cda_module: boolean;
    attribute cda_module of arithmetic_unit: component is TRUE;
begin ...
```

These lines of code perform two tasks. First is to declare *cda_module* as a boolean type of attribute. Second is to apply the attribute to the component *arithmetic_unit*. For any other components that need the *cda_module* attribute, all that is required is the second attribute statement which applies the attribute to the component. In this fashion, the layout can be preserved for some components and destroyed for others as needed in the same architecture.

### 5.1.3 Signal-to-pad mapping

After all the hierarchy has been established and the physical-level partitioning is complete, then all the signals that flow in and out of the top-level entity should be known. Some of the signals will be connected to perimeter pads and some will be connected to area pads. It is at this point that the designer must decide which area-pad signals are connected to which area pads. This step involves assigning pad numbers to each signal based on knowledge of the area-pad array layout. For example, if the array were like the one shown in Figure 5-3, then the signals would have to be assigned to pads that are signal

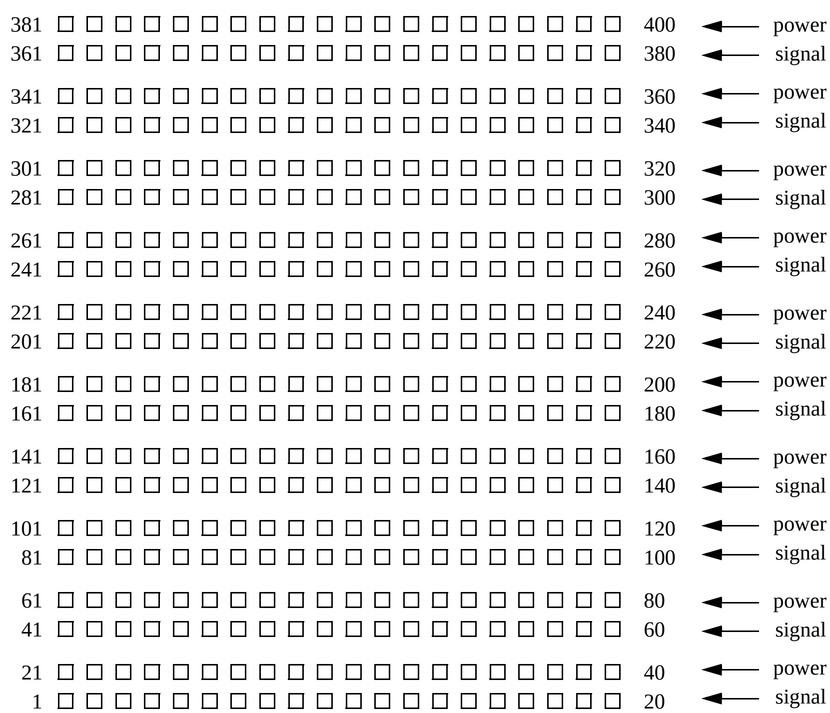


Figure 5-3. Area-pad array with numbers

pads and they would skip over the numbers assigned to power pads. Thus, a list should be created that lists all the signals and the pads they are assigned to such as the one in Table 5-1. From the table, the signals and their pad numbers can be entered into the code gener-

| *Signal name* | *Direction* | *# of pads* | *Assigned pad numbers* |
|---|---|---|---|
| adder_a | I | 32 | 1-20, 41-52 |
| adder_b | I | 32 | 53-60, 81-100, 121-124 |
| mult_a | I | 32 | 125-140, 161-176 |
| mult_b | I | 32 | 177-180, 201-220, 241-248 |
| prod | O | 64 | 249-260, 281-300, 321-340, 361-372 |

Table 5-1. Assigning signals to pads

ator program.

### 5.1.4 Area-pad signal connections

When the circuit is being designed and block diagrams are being drawn, signals at the top level are drawn as if they go straight from the outside world into the logic. But, there are actually a number of different wires that have to be connected before the signal reaches the logic. The logical signal has to be connected to an area pad and the output of the area pad connects to the receiver which then feeds the signal to the circuitry for an input as shown in Figure 5-4 (with signal names from Table 5-1). On the output side, the signal travels from the output of the logic gate through a buffer before reaching the area pad as shown in Figure 5-5. This is the same architecture for every single bit connection in the design that is connected to area pads for both inputs and outputs. In addition, every one of the connections that is given a signal name in the figures has to be accounted for in the VHDL source code. The signals that are the inputs and outputs of the top-level block, such as *adder_a* and *prod*, are listed in the entity declaration. The other signals that are con-

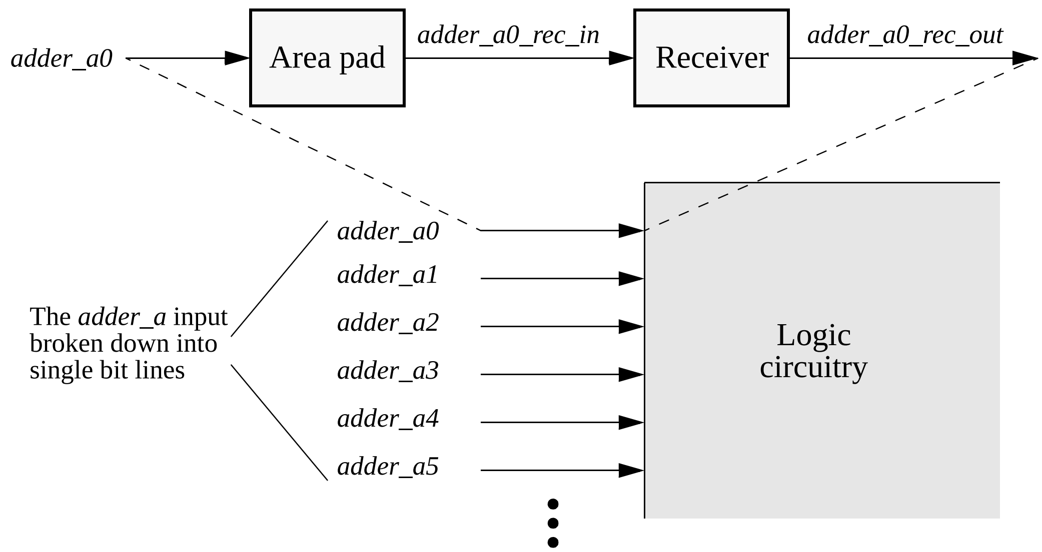


Figure 5-4. A block schematic showing the input connections to an area pad

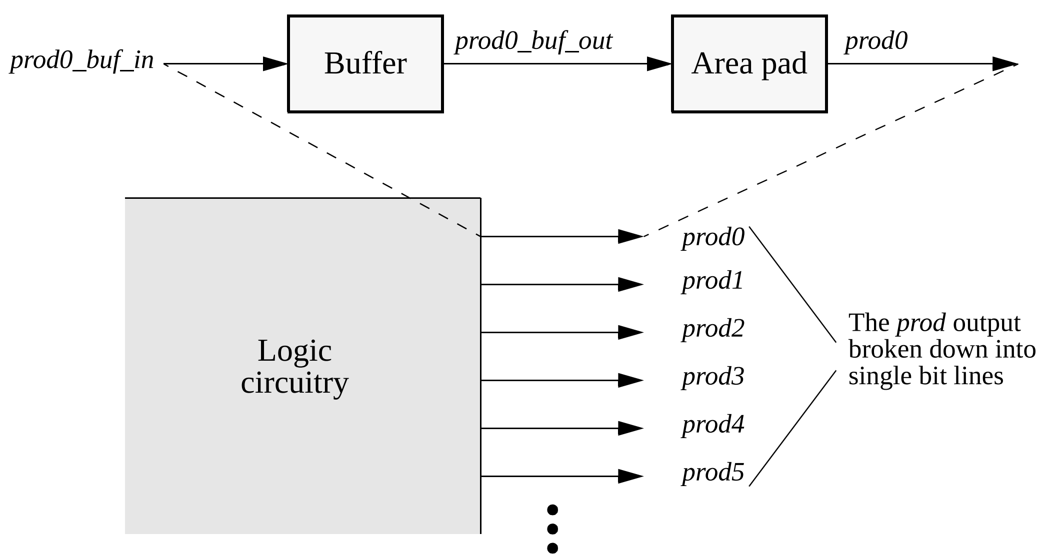


Figure 5-5. A block schematic showing the output connections to an area pad

nected to the receivers and buffers are listed in the architecture in the signal declaration portion before the *begin* statement. Fortunately, each one of the bit signals does not have to be listed individually. The *std_logic_vector* data type includes all of the bits.

### 5.1.5 Area-pad attributes

There are special attributes that have to be added to the source code for the area pads. The first one is the attribute that tells Epoch which package to select. This is the package from the package file that is discussed in Section 4.2.2. The package name is declared in the top-level entity. The syntax is as follows:

```
entity TOP
    port( ...
    );
    attribute PACKAGE_NAME : string;
    attribute PACKAGE_NAME of TOP : entity is “area_pad_package”;
end TOP;
```

The *PACKAGE_NAME* is a string attribute. Its value is the name of the *.pkg* file that was created for the area-distributed I/O padframe. In this case, the file would have been called *area_pad_package.pkg*.

The other attributes apply specifically to the pads. One tells Epoch the pin number of the pad, so that Epoch will know where to position the pad based on the number and the corresponding coordinates it finds in the package file. The second tells Epoch what type of pad each one is. The syntax is as follows:

attribute PINNUM of *pad_label*: label is *pad_number*;

attribute PAD of *pad_label*: label is “*pad_type*”;

The *pad_label* field of both attributes refers to the label that each pad is given when it is instanced. The *pad_number* is the actual number of the pad. The *pad_type* is one of three possible pad types -- Vdd, GND, or PAD. The PAD type is for signal lines. An example of these attributes follows:

attribute PINNUM of IN_PAD1: label is 1;

attribute PAD of IN_PAD1: label is “PAD”;

5.1.6 Instancing the area pads

The area pads (both input and output) are components with two ports. They have an input port and an output port, as shown in Figure 5-6. So signals need to be attached to

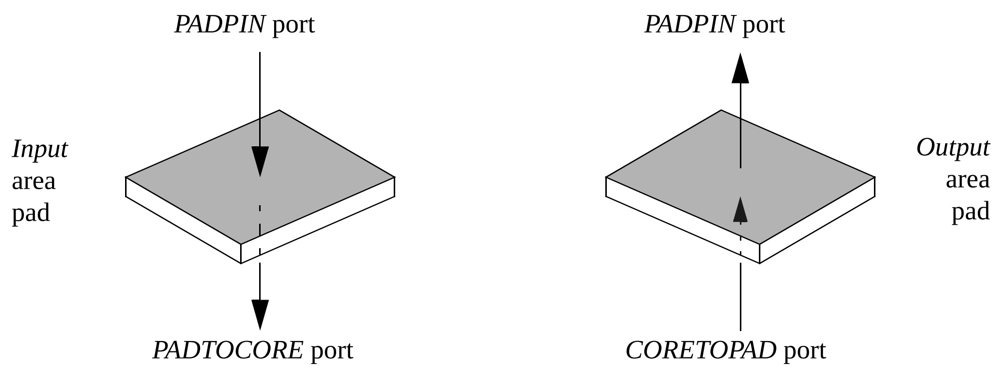


Figure 5-6. Area pad ports

them in the VHDL structural source code. The following is an example of the syntax for an input area pad:

IN_PAD0: areapadin

port map (PADPIN=>mult_in1_opto(0), padtocore=>mult_in1_opto_int(0));

The signal connected to the *PADPIN* port is an entity-level signal. The signal connected to the *padtocore* port is an internal signal.

5.1.7 Source code generator

For each area pad, four lines of VHDL source code must be written. They are the pin number attribute (*PINNUM*), the pad type attribute (*PAD*), and the two lines that instance the area pad. In addition, for each signal pad, either a receiver or a buffer has to be instanced. So six lines of code have to be written for some of the pads. All the interconnecting signals have to be listed in the code as well. This situation leads to an extremely voluminous amount of source code that is very repetitive to be written. Therefore, a program to generate this tedious code is necessary. The program should read in the number and names of the input signals, the number and names of the output signals, the numbers of the pads that each signal is connected to (as from Section 5.1.3), and any pads that are to be left unconnected. Even if there is no logical signal connected to a pad, it still has to be instanced and connected to a dummy signal in the source code because all of the area pad array has to be placed so that the optical device wafer will stick to the silicon wafer during the bonding process.

The program should produce all the attributes and instances that are necessary for each pad. It should also produce signals that should go in the port list of the entity declaration and the internal signals that go in the architecture that connect the area pads to the receivers and buffers. An example of a source code generator is listed in Appendix 3.

## 5.2 CAD tool executables

The physical design tools for area pad designs need to operate at two levels: the main circuitry of the design (also called the "core") and at the area pad level of hierarchy. These two areas are separate, yet interdependent, of one another.

Using this methodology, the main components of the software operation are the Epoch executables and Eggo router. Epoch includes programs to perform hierarchical placement, routing, and buffer sizing. Epoch completes the physical design of all levels within the subblocks. Afterwards, Eggo places the area pads and routes them to the rest of the design.

There is one more setup step to perform before proceeding with the physical design process. If this design uses either the pixel-array or over-active-circuitry smart pixel layout styles, then the area pads will be placed over active logic core-level circuits. Therefore, the core-level circuits cannot be routed with the top layer of metal, because they will short with the area pads, which will be placed in the top layer of metal. So Epoch has to be told to route the core circuits in the lower two layers of metal only. There is a caveat to this statement. If the fabrication process is a 4-layers of metal process, the area pads and their connections will be placed on the fourth layer and the core circuits will be routed on layers: metal1, metal2, and metal3. The exception is that Epoch will not have to be told to route in the lower three layers of metal only. By default, Epoch does not have the ability to route circuits in layers beyond metal3. The route executables would have to be rewritten to handle a fourth or fifth layer of metal. So the following step applies only to 3-metal processes with pixel-array or over-active-circuitry smart pixel layout styles. The *makechip* file should be copied from the *$CASCADE/data* directory into the *vhdl* directory of the current project. Then, it should be edited to add -*lower2layers* to the following executables found in the *makechip* file:

“datapath” :dpath5 -lower2layers $(CDS_OPTS) $(MSGFILE)/dpath5.log %name

geo2glue:geo2glue -lower2layers $(CDS_OPTS) $(MSGFILE)/geo2glue.log %name

“glue” :gluenp -lower2layers $(CDS_OPTS) $(MSGFILE)/gluenp.log %name

“ferret” :ferret -lower2layers $(CDS_OPTS) $(MSGFILE)/ferret.log %name

“multiplier” :netcells $(CDS_OPTS) $(MSGFILE)/netcells.log %views %force %name; dpath5 -lower2layers -nplace $(CDS_OPTS) $(MSGFILE)/dpath5.log %name

For all designs that have lower level components, then steps 1 through 3 must be performed on the lower level components prior to compiling the top-level entity. If the subcomponents need to have their placement preserved, then they need step 4 also. In addition, they will probably need to be manually floorplanned. The executable to floorplan the layout is: *floorplanner* <design>.

For a design written in VHDL, these steps are performed in Epoch prior to physical design (further help on any of these commands can be found in the Epoch User's Manual [40,41,42]):

1. *vhdlan* <design>.vhd - Checks the VHDL source code for syntax errors and splits the design into an entity and an architecture.
2. *vhdlcomp* <design> - Converts the VHDL structural description into an Epoch netlist and performs logic synthesis as required on very simple behavioral code.

Because of a bug in the Epoch software, the *netinput* executable expects the port name, PADPIN, to be in capital letters when it reads the <design>*.net* file. However, *vhdlcomp*, which produces the <design>*.net* file, prints padpin in lower-case letters. So, all the instances of the word padpin need to be changed to upper-case. The <design>*.net* file is located in the project/<design> directory.

3. *netinput* <design> - Imports a netlist into the Epoch environment and flattens the hierarchy into subblocks.

At this point in the physical design process, it is necessary to assist the placement tool (geo2glue) in placing the buffers, both receivers and transmitters, close to their associated area pads. Close placement is important because the signal between the area pad and the receiver is a low-amplitude current that could be corrupted in the presence of noise that it would encounter in travelling near other high-speed switching circuits. To affect the placement of the buffers, the weight of the net (or connection) between the buffer and the area pad will be increased. Those nets that have a greater weight have a higher in getting the two components that they connect to be placed closer to each other. So a text file needs to be created that contains the names of all the nets that connect area pads to buffers and the maximum weight for a net. The syntax of the text file follows:

ATTRNET_FLOAT *net_name* NET_WEIGHT 5.0

This statement has to be instanced for every net between a pad and a buffer, where the signal name is typed in the *net_name* field. This text file is created for the top-level entity only and is called <top_level_entity>*_core.attr*. The file has to be placed in the <project>/*layout_parts/*<top_level_entity>*_core* directory. After the file is properly stored then the set attribute program has to be run on the design from the *vhdl* directory, like so:

*setattr* <top_level_design>*_core*

This step has to be performed every time *netinput* is executed for the design. Then, the rest of the steps are followed as described below.

The steps for physical design (placement and routing) depend on the number of levels of hierarchy in the design that are attached to area pads. It is recommended that there be only one level of hierarchy for the area pads. If there is only one level of hierarchy that connects to the area pads, then the only physical design step to perform is:

4. *autocompile* <design>_core

This will perform placement, routing, and buffer sizing on all the core circuitry. The design is then ready for area pad routing, step 5.

If the design has multiple levels of hierarchy that connect to area pads, then the *autocompile* command has to be split up into different steps, as follows:

4a. *geo2glue* <design>_core - Develops an initial placement expression for the design.

4b. *floorplanner* <design>_core - An optional step that can change soft groups to hard groups [43].

4c. *hpp -locate* <design>_core - Takes the *h*ierarchical *p*in *p*lacement information from the <design>_core to the next levels down. The pin positioning works in a top-down manner and ripples pin information to the next level of hierarchy below. Therefore, *hpp* needs to be performed on each level of hierarchy in the design.

4d. *geo2glue* <subblock> - Places each subblock of the design with new knowledge of area pad locations. This must be performed on every subblocks.

4e. *autocompile* <design>_core - Routes and buffer sizes the design without changing the already established placement.

The two steps, *hpp* and *geo2glue*, need to be done in conjunction with each other. Once the placement is done at a certain hierarchical level, detailed information can be passed down to the next level of hierarchy. Once the lower-level subblocks are placed, then the upper-levels can achieve a better placement estimation. This is a top-down / bottom-up design flow. Detailed placement of the area pad buffers depends on both high-level goals (precise area pad positioning) and low-level goals (subblock size optimization).

This concludes the Epoch phase of the physical design. The Eggo portion is a simple command:

5. *eggo* <design>

Eggo places the area pads above the design and then completes the routing of the design to the area pads. Sometimes this area pad routing cannot be completed. Often this is the case of over-congestion or poor placement of the associated buffer cells. At this point, it is advisable to return to Step 4a or 4b and iterate.

Most foundries require that the data for the layout be broken down into a series of masks for the different layers of contacts, polysilicon, and metals. The format of that data is either Caltech intermediate format (CIF) or GDS. Epoch has the ability to produce the layout in either type of format. To produce CIF data, type the following in the *vhdl* directory:

*geo2cif -W* <design>

The *-W* option places the CIF file in the current directory. To create a GDS file, type the following in the *vhdl* directory:

*geo2gds* <design>

There are a number of options that work with this command. Type *geo2gds -help* to review them.

## 5.3 Simulation

The design should of course be simulated after layout and before fabrication. However, since the layout contains analog and other imported circuitry, it requires mixed-mode simulation. In the VHDL environment, it is difficult to perform mixed-mode simulation. Therefore, the design has to be simulated piecewise. The receivers and any other imported circuits can be simulated with SPICE or a SPICE-type simulator before being imported into the Epoch database to verify proper performance. In order to verify that the digital logic is working properly, layout has to be implemented without any of the imported cells. Then a VHDL file is produced so that the circuit can be checked in a VHDL simulator.

The steps to follow to create a VHDL output file are:

10. Run the Epoch gui by typing in the *vhdl* directory: *epoch*
11. Click on the *Simulation* pull-down menu and select *Parameters*...
12. Select the appropriate simulation parameters for the PMOS delays and NMOS delays to give a best-case, worst-case, or typical simulation and click on SAVE.
13. Click on the *Simulation* pull-down menu again and select *Simulation output* Á *VHDL*....
14. Select the entity name from the *Existing designs* window. Type in the name of the output simulation file in the *Output File Name* field. Double check the output directory. Click Yes on the *Create Delay File* option.
15. Click Run.

This will create two files -- one that contains all of the gates and models of the interconnects of the layout, and one that contains the timing delays. The two files should be analyzed with the Synopsys program *vhdlan* -- first, the delay file and second, the circuit file. Next, the “testbench” file should be analyzed and the configuration file, if necessary. Then the design should be simulated using either the simulation tool, *vhdlsim*, or the dbx tool, *vhdldbx*.

### 5.3.1 Connectivity check

Since the entire layout cannot be simulated at the same time, there is one final check that can be run and definitely should be run. Connectivity of some critical signals and power-to-ground shorts should be checked. A CIF version of the layout should be produced either from Epoch or L-Edit. L-Edit is preferable because unnecessary layers can be turned off and the hierarchy of the design can be flattened. Only the layers of metal, the vias, polysilicon, and the poly contacts are needed in order to make the check run faster.

The layout editor, *magic*, is used to check the connections. Before executing *magic*, the proper technology file has to be linked. The CIF input styles of each technology file have to be checked until the style with the correct lambda units is found. That technology file is loaded when *magic* is invoked, like so:

*magic -dx11 -T* <technology_file>

Once *magic* is loaded, there are just a few simple commands that need to be issued. First the design rule checker must be disabled by typing:

*:drc off*

The CIF input style should be loaded by typing:

*:cif istyle* <istyle_name>

The CIF file should be read by typing:

*:cif read* <file_name>

Once loaded, the design can be viewed by typing *v*, moving the mouse pointer over the outline of the top cell and typing *s*, and finally by typing *x*. Once the layout is fully expanded, simply move the mouse pointer over the signal to be checked and type *s* three times. Repeat as desired. When finished checking, type *:quit* and don't save any of the cells.

## CHAPTER 6 - ANALYSIS AND CONCLUSIONS

### 6.1 Design example

In order to verify the methodology presented in this dissertation, the process was applied to the design and fabrication of an optoelectronic VLSI circuit. The circuit is a multiply-accumulator (MAC), whose block diagram is shown in Figure 6-1, which is

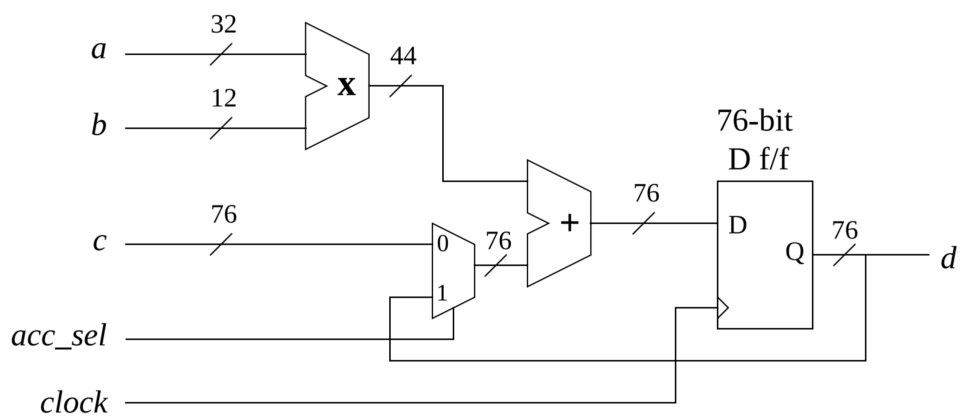


Figure 6-1. Multiply-accumulator block diagram

commonly used in DSP applications. The inputs *a* and *b* are multiplied and added to either input *c* or the previous result, depending on the state of the accumulator select line, *acc_sel*. If *acc_sel* is a ‘0’, the *a**b* product is added to *c*. If *acc_sel* is a ‘1’, then *a**b* is added to the previous result. The adder and the multiplier were synthesized from dataflow VHDL using Synopsys’ dc_shell. The MAC component was then instantiated at the top level with the area pads. The source code for this circuit is shown in Appendix 1. Layout at the top level was performed following the steps from Chapter 5. The layout is shown in

Figure 6-2. The dimensions of the IC are 2141µm x 2141µm. There are 21,000 transistors.

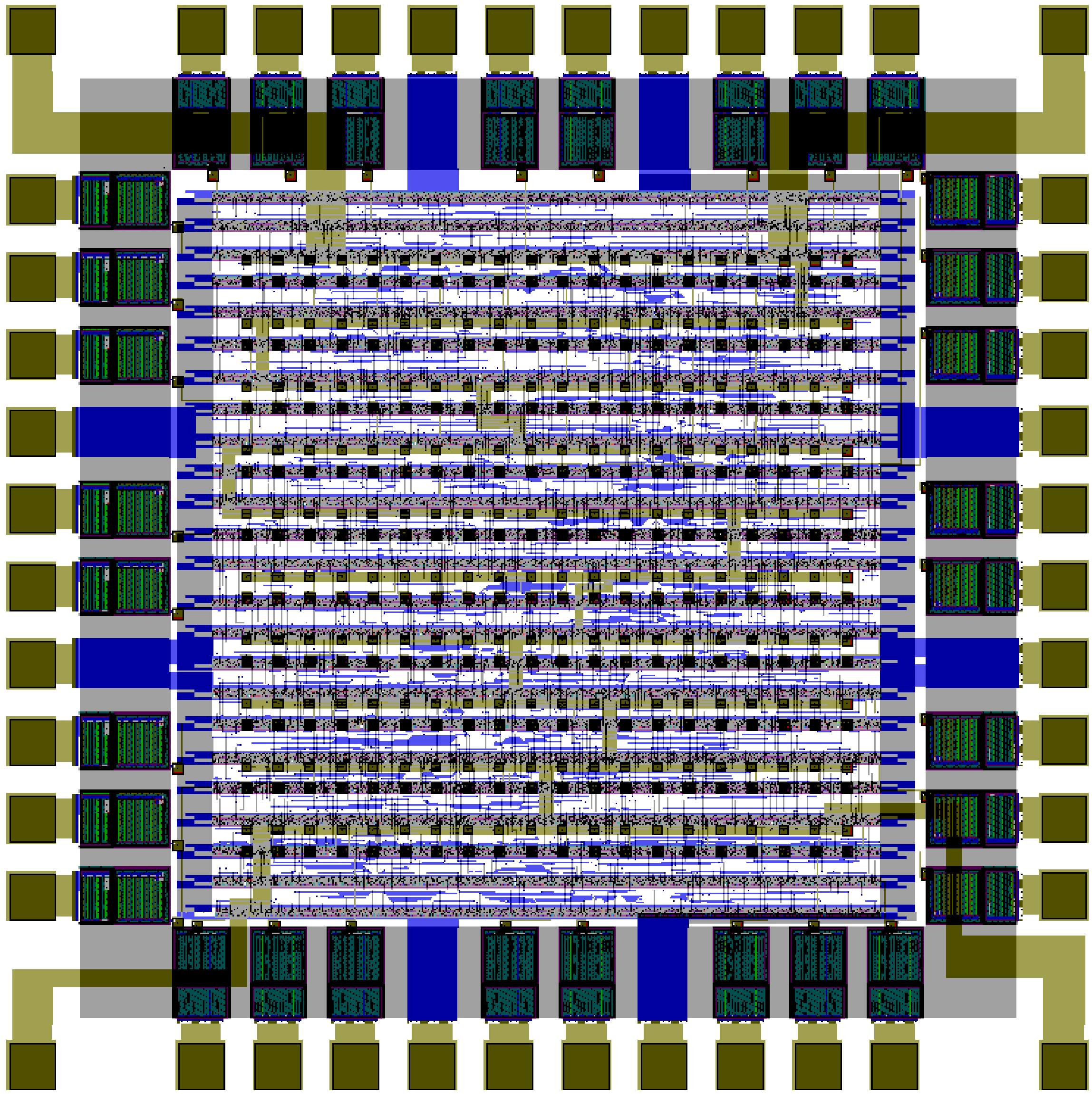

Figure 6-2. Layout of the multiply-accumulator

There are 44 perimeter I/O pads and 400 area pads to yield 200 optical I/Os. Layout and fabrication were performed in Hewlett-Packard’s 3.3V 0.5µm process. The optical devices were added as described in Section 2.2.2 at Bell Labs in Holmdel, New Jersey as part of their 1997 MQW foundry workshop.

## 6.2 Analysis of methodology effectiveness

### 6.2.1 Previous layout techniques

Some of the layout styles are discussed in Section 3.1 in the coverage of smart pixels. But that section doesn't talk about how the layout was actually performed. For the pixel array, one pixel was placed and routed manually that included the power and ground rails such that the pixel could be arrayed and the power supply rails would lock together. Then, the pixel array would be aligned with the area pads which already had power routed to them.

The over-active-circuitry and PIM styles are somewhat similar. The core circuitry was completely placed and routed using Epoch with two long columns of metal3 (top layer of metal) stubs that were placed on the two sides of the layout, as shown in Figure 6-3. The

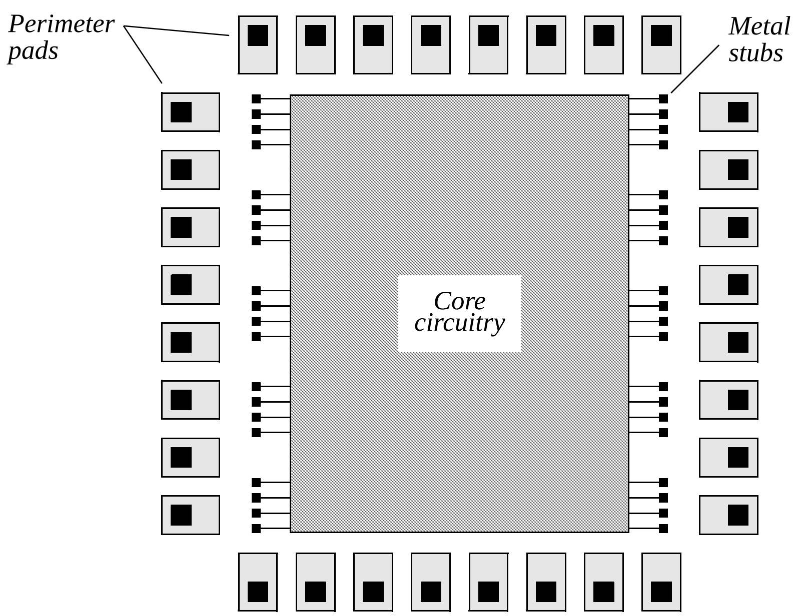


Figure 6-3. Layout model with metal stubs

metal stubs were connected to the inputs of receivers and the outputs of transmitters. Each column was drawn in L-Edit and imported as a block of geometry with port locations at each stub that were routed in metal2 and metal1 to the core (shown in Figure 6-3), such that the only part of the core circuit layout that was in metal3 were the stubs. Separately, a mask (completely in metal3) of the area pads was created in L-Edit, with each of the pads either connected to a power rail or to a signal wire that was routed to a stub at the edge of the array as shown in Figure 6-4. Then, the area-pad mask and the core circuit with the

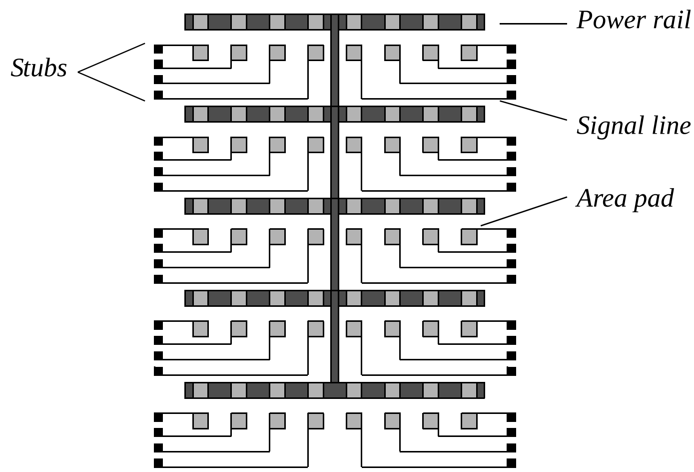


Figure 6-4. Area-pad mask

stubs were imported into L-Edit. The area-pad mask was placed over the core circuit so that the stubs of the area pad mask were aligned with the stubs of the core. The end result is shown in Figure 6-5, the 1-kbit SRAM circuit, a typical example of the over-active-circuitry manual layout style. If there were misalignment, the stubs on the area-pad mask could easily be moved in or out until they lined up in the x-direction. But, if there were a shift in the y-direction in the column of stubs on one side of the core circuit with respect to the column on the other side, then the core circuit would have to be manually floorplanned with the Epoch program, *floorplanner*, then automatically re-routed, and imported into L-Edit again, where hopefully the stubs would line up.

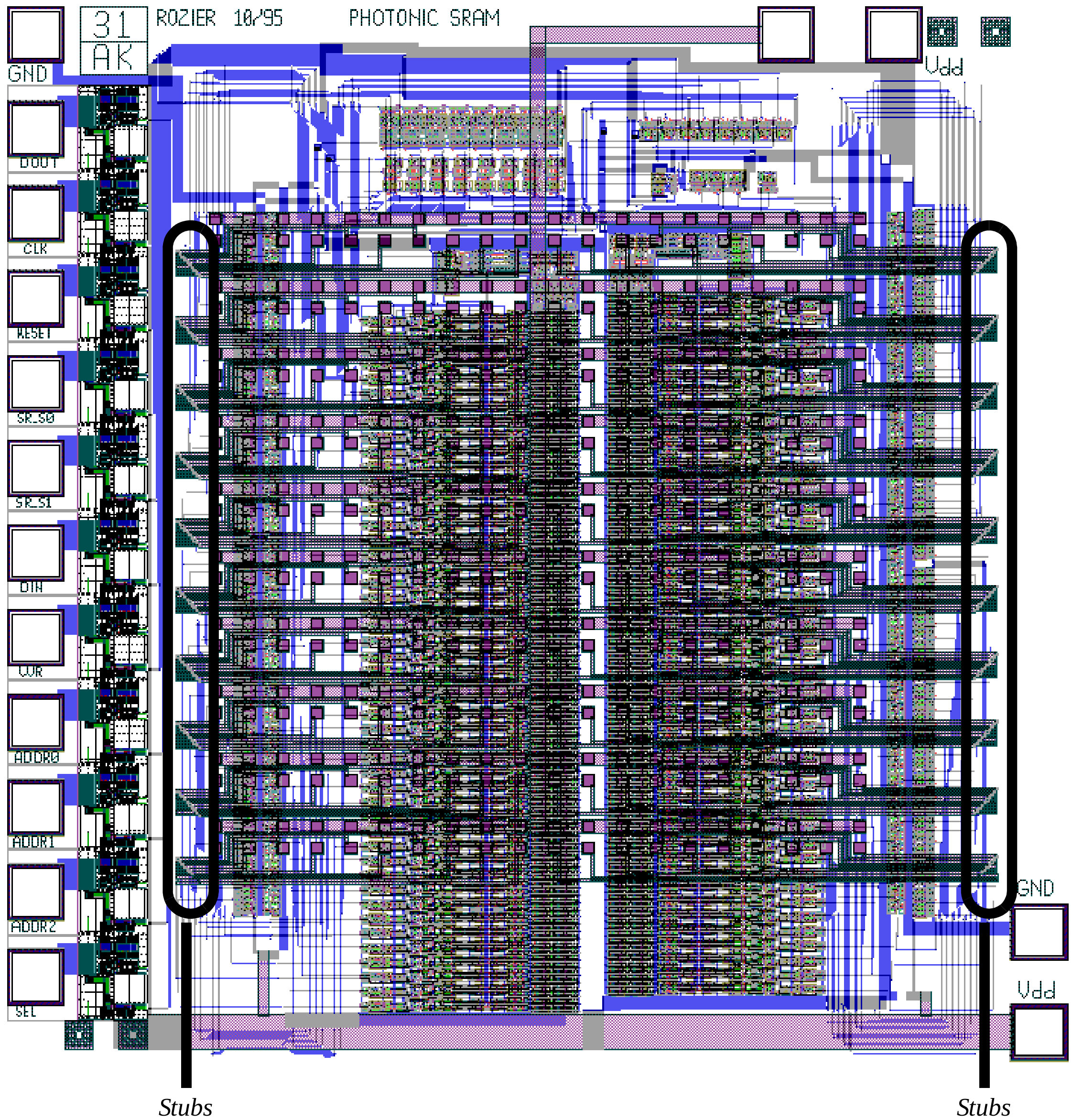


Figure 6-5. Layout of an optoelectronic circuit showing alignment stubs

### 6.2.2 Comparison of methodologies

The manual mask-alignment technique produces a very regular layout. The circuit of Figure 6-5 with only metal3 and glasscuts visible is shown in Figure 6-6. The signals are routed from the pads to the edge where the stubs are located as shown in the zoomed-in box of Figure 6-6. From there, the signals are wired to the receivers and transmitters in

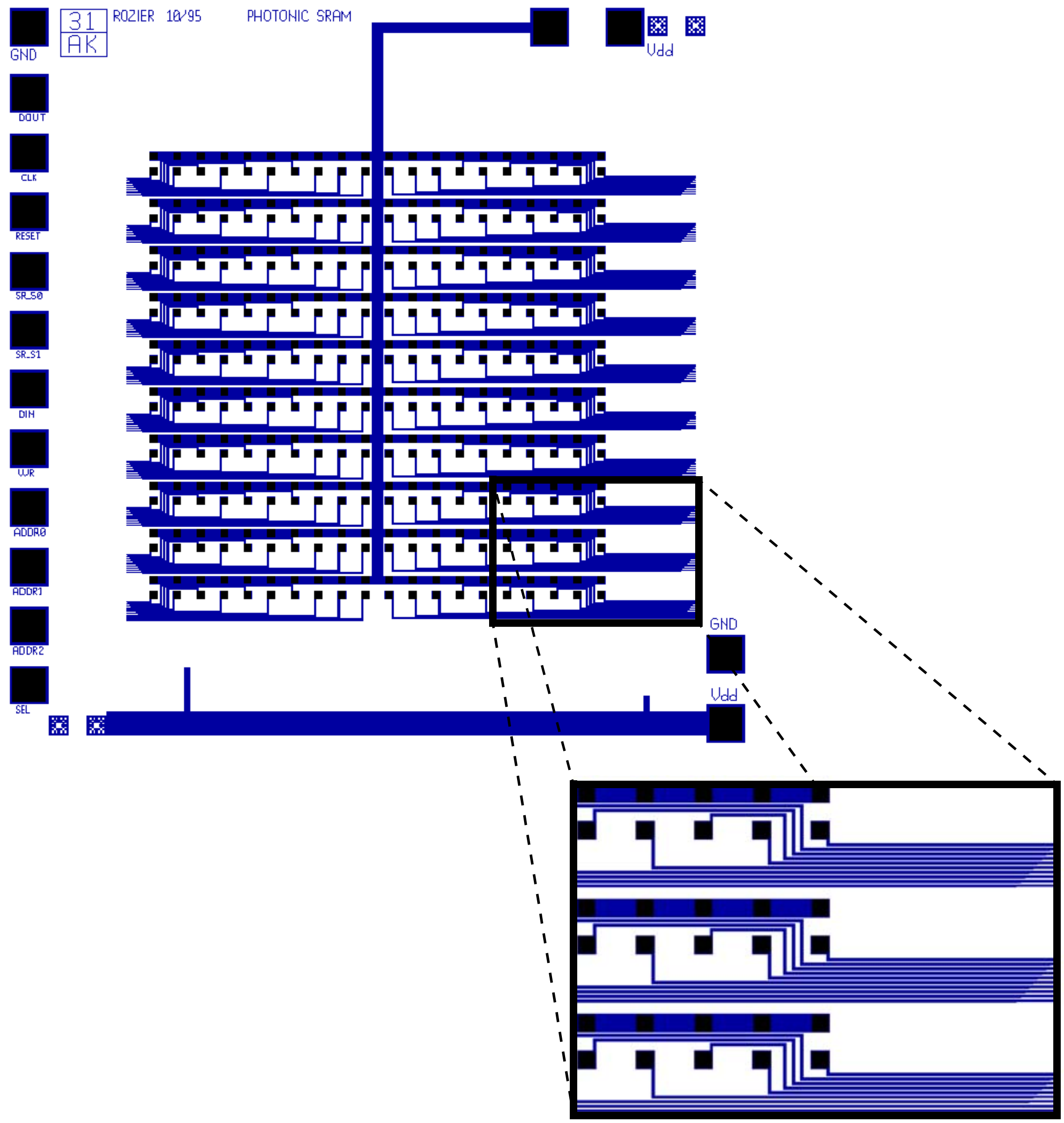


Figure 6-6. Close-up of manual layout

metal2 and metal1.

Figure 6-7 shows the layout with metal3 and glasscuts only of the MAC circuit from Figure 6-2. The layout was performed mostly by computer with manual editing to connect the power rails from the area pads to the perimeter pads in the corners. However, that

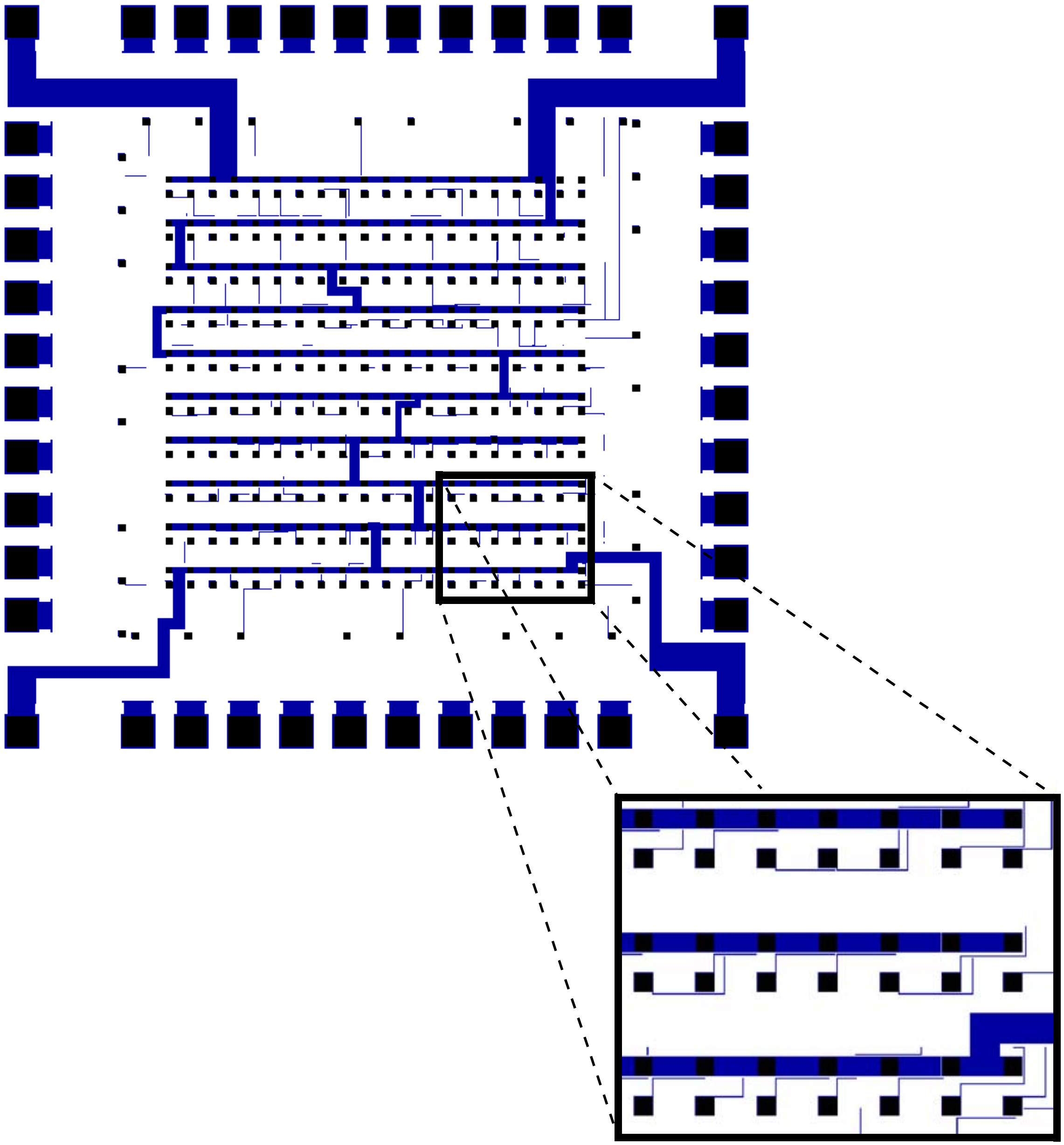

Figure 6-7. Close-up of automated layout

could be performed by the software by importing the connections to the perimeter pads as discussed in Section 4.2.1.

It is desired to keep the wiring length from a pad to its associated buffer (either receiver or transmitter) at a minimum. This is because the signal from the detector to the receiver is a low-amplitude current that can be easily corrupted in the presence of noise. The noise can cause the output of the receiver to falsely switch from a ‘0’ to a ‘1’, or vice-versa, and thus cause an error in the data. In some cases with the manual mask-alignment method, all the receivers and all the transmitters are combined in two blocks that can be placed near the columns of stubs. This is the best situation for keeping the wiring length from the pad to its associated buffer minimized. If the receivers and transmitters are mixed in with the rest of the layout, then these wiring lengths will increase because the wire would have to run from the area pad out to the metal3 stub down to metal2 or metal1 and back to the center of the layout, thereby degrading the performance of the circuit. As seen from Figure 6-7, the layout from the automated method is very irregular. This is due to the fact that the buffers are placed in the sea of standard cells as close to their associated area pads as possible. Contrasted with the manual style, this is a benefit because in the automatic layout style, the wires don’t have to run out to the edge of the core to meet some wiring stubs and then come back in. The connections are routed much more directly.

Table 6-1 shows the statistics of routing lengths from pad to buffer in metal3 for the

| *Layout style* | *Minimum routing length* | *Average routing length* | *Maximum routing length* |
|---|---|---|---|
| Manual | 250μm | 500μm | 850μm |
| Automatic | 0μm | 100μm | 280μm |

Table 6-1. Comparison of the wiring lengths of manual and automated techniques

manual layout technique and the automated technique which is the focus of this dissertation. The numbers for the manual style do not include the length of routing in metal2 and metal1. These numbers were derived by loading the two designs into L-Edit and actually measuring the lengths of a number of traces that connect to area pads. It was easy to measure with the manual technique. The minimum routing length was the distance from the area pad closest to the edge to the stub. The average length was the distance from the middle pad on the right side to the stub, and the maximum length was from the pad closest to the center out to the stub. In quite a few cases with the automatic technique, the buffer was placed underneath the area pad, giving a zero routing length. Most of the traces were around the average length of 100μm. And, only a few were over 200μm.

## 6.3 Conclusions

A means of designing optoelectronic ICs for use with free-space optical interconnects was needed that takes into account the special issues that are associated with optoelectronics. This dissertation attempts to provide that methodology. Since FSOI implies that the design or system will be on more than one chip, system partitioning is important. Putting the appropriate functionality on each chip while maintaining the timing of the system is critical. Taking advantage of the bandwidth improvement that optical interconnects offer should be part of the process. Finally, being able to define and enter the design in a flexible manner such as through a high-level hardware description language like VHDL and automate the physical design process are also necessary.

Compared with conventional methods, this approach focuses most of the design work on the creation of the source code rather than on manual transistor layout. For this reason, this methodology has a shorter design time cycle and is able to handle changes to the design at any phase more easily than manual methods. The physical design preparation activities, such as creating the area pad package file, are one-time investments, so they do not add to the total cost of each design. With the manual layout methods, there is the possibility that each area-pad mask has to be designed separately. Obviously, this would be more time-consuming because each mask could take a couple of hours to produce. By comparison, Eggo placed the area pads and routed their connections for the design example in 2 minutes and 30 seconds on an UltraSparc with 320MB of RAM. Even with a slower machine the automated process would still be much better than creating the area pad mask by hand. Furthermore, using automated programs is a benefit because the software does a thorough job of checking the design for signals that are shorted or disconnected (a common occurrence in hand layout), that might not have been found while performing manual layout. This is an invaluable bonus because the overwhelming majority of ICs that fail do so not because of errors in the logic but because of electrical shorts and opens. Additionally, there is a problem when, in the manual mask-alignment method, the two columns of metal3 stubs get skewed in the vertical direction. This happens occasionally because Epoch will move the placement of various blocks around in order to accommodate routing channels or to minimize the size of the layout. When this happens, the stubs and the signal wires on the area-pad mask could possibly be shifted vertically in order to align the area-pad-mask stubs with the core-circuitry stubs, but there is a limit to the distance that the signal wires can travel in the vertical direction before the wires from one channel intersect the wires from another channel, causing a short circuit. The less risky option is to go back to the Epoch floorplanner and reposition the column of stubs in the core circuit, recompile the layout, and hope that the micro-placement process that Epoch does to create routing channels did not move the stubs too far. This situation clearly illustrates the need for an automated physical design process. Fortunately, as far as manual layout goes, the methodology presented in this dissertation is a “hands-off” process.

Generally, when a process is automated or a task gets accomplished more quickly, there is a price to pay in the quality of the result. The trade-off is that if the job is done faster, then the result is lower in quality. However, with this methodology, just the opposite is true. In this case, the quality of the layout is judged by the closeness of the buffers to their associated area pads. The highest performance with flip-chip ICs is achieved by placing the pads and their buffers close to one another, thereby keeping the wiring lengths to a minimum. As seen in Table 6-1, on the average, the automated layout was five times better than the manual layout. And, for the most part, the longest traces with the automated method were shorter than the shortest manual traces.

There are a couple of drawbacks to the VLSI physical design of this methodology. One is the general problem that is inherent with flip-chip bonding. The top layer of metal has to be dedicated to the area pads and the connections to the area pads. So the core circuitry has to be routed in the lower layers of metal, which increases the size of the layout because the size of the routing channels is reduced with more layers of routing metal. Larger layout area obviously leads to greater cost. The other drawback to this approach is that it in a three metal-layer process, metal1 and metal2 are heavily used in the underlying core circuitry. There is not much room to complete the area-pad signals in metal2 after the core is done. Eggo does not presently rip-up & re-route any of the core circuitry wiring that has already been done. Problems could occur. It is possible that in routing the core, a port that needs to connect to an area pad could be surrounded on all four sides by metal2. Also, in routing other area-pad signals, the area-pad router may have already surrounded the same port with metal3. In both cases, the port may be stranded so that the router cannot connect to it in any combination of metal2 and metal3, and an unrouted net results. For this reason, it is critical to have the buffers that connect to the pads be placed very closely to the intended pad. The probability that a net will be unrouted increases as the distance from the buffer to the area pad increases. Also, the likelihood of unroutability would increase with the number of area pads in a design. In summary, area-pad physical design does not begin at the final routing stage; it needs to occur throughout the placement of the design's subblocks. On the other hand, these problems can be reduced by utilizing four-layer (or more) metal fabrication processes.

Another minor problem of the approach being presented is the difficulty of the size of the HDL netlist. Four lines of VHDL code are needed for each area pad, meaning that typing the code by hand for any number greater than 10 pads would be impractical. However, a well-written netlist generator program, as discussed in Section 5.1.7, could be used for a variety of designs.

This methodology is excellent for an area-pad IC of moderate size -- around 400 area pads and less than 100k transistors. This technique has not been used on larger designs but should easily scale up to circuits with more than 2000 area pads.

For the future, this methodology could be improved. It would be nice for the CAD tools to be more integrated. If the area pads could be placed and routed at the same time as the core circuitry, then the core could be routed to a certain extent in the top layer of metal. This would reduce the size of the circuitry and reduce or eliminate the number of unrouted area-pad nets. With the simultaneous core circuit and area-pad placement should come the ability of the placement tool to recognize the relationship between an area pad and its buffer so that the two would be placed close to each other without having to explicitly perform this activity via the set attribute (*setattr*) function. It would also be nice for the *net-*

*input* program to recognize the PADPIN port in the netlist without it having to be capitalized. This would eliminate some tedious and unnecessary text editing. Finally, mixed-mode, full-scale simulation of the circuit after layout would be highly desirable.

# REFERENCES

[1] T. K. Woodward, A. V. Krishnamoorthy, A. L. Lentine, L. M. F. Chirovsky, “Optical Receivers for Optoelectronic VLSI”, *IEEE Journal of Selected Topics in Quantum Electronics,* vol. 2, no. 1, April, 1996, pp. 106-116.

[2] A. L. Lentine, K. W. Goossen, J. A. Walker, L. M. F. Chirovsky, L. A. D’Asaro, S. P. Hui, B. J. Tseng, R. E. Leibenguth, J. E. Cunningham, W. Y. Jan, J. M. Kuo, D. W. Dahringer, D. P. Kossives, D. D. Bacon, G. Livescu, R. L. Morrison, R. A. Novotny, D. B. Buchholz, “High-speed optoelectronic VLSI switching chip with >4000 optical I/O based on flip-chip bonding of MQW modulators and detectors to silicon CMOS”, *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 2, no. 1, April, 1996, pp. 77-84.

[3] Digital Equipment Corporation web site, http://www.digital.com/semiconductor/alpha/news/press02feb.htm, DEC, Boston, MA, May 25, 1998.

[4] MIDAS service (from MOSIS) web site, http://www.isi.edu/midas/, ISI, Marina Del Ray, CA, May 25, 1998.

[5] Intel web site, http://www.intel.com/procs/ppro/info/techdtl/p6pkg.htm, Intel, Santa Clara, CA, May 25, 1998.

[6] MPPOI’96, *Proceedings of the Third International Conference on Massively Parallel Processing using Optical Interconnects*, IEEE Computer Society, Gottlieb, A., Li, Y., Schenfeld, E., editors, Los Alamitos, CA, Oct. 27-29, 1996.

[7] Forrest, S. R. and Hinton, H. S., editors, Special issue on smart pixels. IEEE J. Quantum Electronics, vol. 29, no. 2, March 1993.

[8] Smart Pixels ‘96, *Digest of the IEEE/LEOS 1996 Summer Topical Meetings*, IEEE Cat. Num. 96TH8164, Aug. 5-9, 1996.

[9] Goossen, K. W., Cunningham, J. E., Jan, W. Y., “GaAs 850nm Modulators Solder-Bonded to Silicon”, *IEEE Photonics Technology Letters*, vol. 5 no. 7, July, 1993, pp. 776-778.

[10] Krishnamoorthy, A.V.; Ford, J.E.; Goossen, K.W.; Walker, J.A.; Lentine, A.L.; Hui, S.P.; Tseng, B.; Chirovsky, L.M.F.; Leibenguth, R.; Kossives, D.; Dahringer, D.; DAsaro, L.A.; Kiamilev, F.; Aplin, G.F.; Rozier, R.G.; Miller, D.A.B.; “Photonic page buffer based on GaAs multiple-quantum-well modulators bonded directly over active silicon complementary-metal oxide-semiconductor (CMOS) circuits”, *Applied Optics*, vol. 35, no. 14, May 10, 1996, pp. 2438-48.

[11] R. Pu, E. M. Hayes, R. Jurrat, C. W. Wilmsen, K. D. Choquette, H. Q. Hou, K. M.

Geib, “VCSELs Bonded Directly to Foundry Fabricated GaAs Smart-Pixel Arrays”, *IEEE Photonics Technology Letters*, vol. 9, no. 12, December, 1997, pp. 1622-1624.

[12] Jewell, J. L., Harbison, J. P., Scherer, A., Lee, Y. H., Florez, L. T., “Vertical-Cavity Surface-Emitting Lasers: Design, Growth, Fabrication, Characterization”, *IEEE Journal of Quantum Electronics*, vol. 27 no. 6, June 1991, pp. 1332-1346.

[13] Geels, R. S., Corzine, S. W., Scott, J. W., Young, D. B., Coldren, L. A., “Low Threshold Planarized Vertical-Cavity Surface-Emitting Lasers”, *IEEE Photonics Technology Letters*, vol. 2 no. 4, April, 1990, pp. 234-236.

[14] MacDougal, M. H., Dapkus, P. D., Pudikov, V., Hammin, Z., Yang, G. M., “Ultralow Threshold Current Vertical-Cavity Surface-Emitting Lasers with AlAs Oxide-GaAs Distributed Bragg Reflectors”, *IEEE Photonics Technology Letters*, vol. 7 no. 3, March, 1995, pp. 229-231.

[15] T. J. Cloonan, G. W. Richards, A. L. Lentine, F. B. McCormick, H. S. Hinton, S. J. Hinterlong, “A Complexity Analysis of Smart Pixel Switching Nodes for Photonic Extended Generalized Shuffle Switching Networks”, *IEEE Journal of Quantum Electronics*, vol. 29, no. 2, February, 1993, pp. 619-634.

[16] MOSIS web site, http://www.mosis.org/cmos-packages.html, ISI, Marina Del Ray, CA, May 25, 1998.

[17] Krishnamoorthy, A. V., Woodward, T. K., Goossen, K. W., Walker, J. A., Lentine, A. L., Chirovsky, L. M. F., Hui, S. P., Tseng, B., Cunningham, J. E., and Jan, W. Y., “Operation of a single-ended 550Mbits/sec, 41-fJ hybrid CMOS/MQW receiver-transmitter”, *Electronics Letters*, vol. 32 no. 8, April, 1996, pp. 764-765.

[18] Intel web site, http://www.intel.com/PentiumII/home.htm, Intel, Santa Clara, CA, May 25, 1998.

[19] J. E. Ford, J. A. Walker, J. E. Cunningham, M. D. Feuer, “Transmissive Fiber Optic Packaging for Reflective Modulators”, Conference Proceedings. LEOS '96 9th Annual Meeting. IEEE Lasers and Electro-Optics Society 1996 Annual Meeting, Nov. 1996, vol. 2, pp. 244-5.

[20] F.E. Kiamilev, R.G. Rozier, and A.V. Krishnamoorthy, “Smart Pixel IC Layout Methodology and its Application to Photonic Page Buffers”, *Intl. Journal of Opto-electronics*, Vol. 11, No. 3, pages 199-216, November 1997.

[21] R. G. Rozier, R. Farbarik, F. E. Kiamilev, J. T. Ekman, P. V. Chandramani, A. V. Krishnamoorthy, and R. Oettel, “Automated Design of ICs with Area-Distributed I/O Pads”, to appear in *Applied Optics* special issue on Optoelectronic CAD tools.

[22] F.E. Kiamilev, J.S. Lambirth, R.G. Rozier, and A.V. Krishnamoorthy, "Design of a

64-bit microprocessor core IC for hybrid CMOS-MQW technology," Proceedings of MPPOI 1996, IEEE Computer Society, Gottlieb, A., Li, Y., Schenfeld, E., editors, Los Alamitos, CA, Oct. 27-29, 1996, pp. 53-60.

[23] J. Ekman, "Hybrid Optoelectronic Programmable Arithmetic Unit HP14TB 0.5µm CMOS-MQW Foundry Run," Internal Technical Report, UNC Charlotte, 1997.

[24] A. V. Krishnamoorthy, J. E. Ford, K. W. Goossen, A. L. Lentine, S. P. Hui, B. Tseng, L. M. F. Chirovsky, R. Leibenguth, D. Kossives, D. Dahringer, L. A. D'Asaro, F. E. Kiamilev, G. F. Aplin, R. G. Rozier, and D. A. B. Miller, “Photonic Page Buffer based on GaAs MQW modulators bonded directly over active silicon CMOS circuits”, *Applied Optics*, Vol. 35, No. 14, pp. 2439-2448, May 1996.

[25] A. V. Krishnamoorthy, R. G. Rozier, J. E. Ford, and F. E. Kiamilev, “CMOS Static RAM Chip with High-Speed Optical Read and Write”, *IEEE Photonics Technology Letters*, vol. 9, no. 11, Nov., 1997, pp. 1517-1519.

[26] K.K. Chau, M.W. Derstine, S. Wakelin, J. Cloonan, F.E. Kiamilev, A.V. Krishnamoorthy, K.W. Goosen, J.A. Walker, J.E. Cunningham, W.Y. Jan, B. Tseng, S. Hui, L.M.F. Chirovsky, "Smart Pixel Memory Buffer Array with Parallel and Serial Access", Digest IEEE/LEOS 1996 Summer Topical Meetings on Smart Pixels, pp. 28-29, (August, 1996) Keystone, Colorado.

[27] A.V. Krishnamoorthy, J.E. Ford, K.W. Goosen, J.A. Walker, B. Tseng, W.Y. Jan, T.K. Woodward, R.G. Rozier, F.E. Kiamilev, and D.A.B. Miller "Fabrication and testing of AMOEBA: an optoelectronic switch for multiprocessor networking", Digest IEEE/LEOS 1996 Summer Topical Meetings on Smart Pixels, pp. 48-49, (August, 1996) Keystone, Colorado.

[28] A.V. Krishnamoorthy, J.E. Ford, K.W. Goosen, J.A. Walker, B. Tseng, S.P. Hui, J.E. Cunningham, W.Y. Jan, T.K. Woodward, M.C. Nuss, R.G. Rozier, F.E. Kiamilev, and D.A.B. Miller, "The AMOEBA chip: an optoelectronic switch for multiprocessor networking using dense-WDM", Proceedings of MPPOI 1996, IEEE Computer Society, Gottlieb, A., Li, Y., Schenfeld, E., editors, Los Alamitos, CA, Oct. 27-29, 1996, pp. 94-100.

[29] Franzon, P. D., Stanaski, A., Tekmen, Y., Banerjia, S., “System Design Optimization for MCM”, IEEE MCM Conference Proceedings, 1995.

[30] Dehkordi, P., Ramamurthi, K., Bouldin, D., Davidson, H., Sandborn, P., “Impact of Packaging Technology on System Partitioning: A Case Study”, IEEE MCM Conference Proceedings, 1995.

[31] Banerjia, S., Glaser, A., Harvatis, C., Lipa, S., Pomerleau, R., Schaffer, T., Stanaski, A., Tekmen, Y., Bilbro, G., Franzon, P., “Issues in Partitioning Integrated Circuits for MCM-D/Flip-Chip Technology”, IEEE MCM Conference Proceedings, 1996.

[32] R. G. Rozier, F. E. Kiamilev, and A. V. Krishnamoorthy, “Design and Partitioning of an FSOI FFT Processor”, TOPS special issue on Spatial Light Modulators, vol. 14.

[33] R. G. Rozier, F. E. Kiamilev, and A. V. Krishnamoorthy, “Design and Evaluation of a Photonic FFT Processor”, *Journal of Parallel and Distributed Computing*, vol. 41, no. 1, Feb. 25, 1997, pp. 131-136.

[34] R. G. Rozier, “The Design of a Parallel High-Speed FFT Processing System”, Master’s thesis, UNC Charlotte, May 1996.

[35] Mazor, S., Langstraat, P., *A Guide to VHDL*, Kluwer Academic Publishers, Boston, MA, 1992.

[36] L-Edit User Manual, Tanner Research, Inc., July, 1996.

[37] *Epoch User’s Manual*, Duet Technologies Corporation, Bellevue, WA, 1997, pp. 5-1 - 5-113.

[38] *Epoch User’s Manual*, Duet Technologies Corporation, Bellevue, WA, 1997, pp. 5-26.

[39] *Epoch User’s Manual*, Duet Technologies Corporation, Bellevue, WA, 1997, pp. 5-42.

[40] *Epoch User’s Manual*, Duet Technologies Corporation, Bellevue, WA, 1997, pp. 7-1 - 7-114.

[41] *Epoch User’s Manual*, Duet Technologies Corporation, Bellevue, WA, 1997, pp. AP2-1 - AP2-32.

[42] *Epoch/VHDL Interface*, Duet Technologies Corporation, Bellevue, WA, 1997.

[43] *Epoch User’s Manual*, Duet Technologies Corporation, Bellevue, WA, 1997, pp. 6-34.

# APPENDIX1:SOURCE CODE FOR MULTIPLY-ACCUMULATE CIRCUIT

```
--------------------------------------------------------------------------
--
--  Copyright 1997 Richard Rozier
--
--  Entity name: mult_32x12_76_chip
--
--  Purpose: Creates a padframe for the CO-OP workshop mult & add chip.
--
--  Author:  Richard Rozier
--
--  Notes: Instances the dw_sram entity, 400 area pads, and 40 perimeter
--                          pads.
--
--  Revision History
--              10/21/97  File started.
--              10/24/97  File finished.
--------------------------------------------------------------------------

library IEEE;
library EPOCH_LIB;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use EPOCH_LIB.COMPONENTS.all;
use EPOCH_LIB.STANDARD_COMPONENTS.all;
use EPOCH_LIB.DPCELLS_COMPONENTS.all;

entity mult_32x12_76_chip is
port(
            VDD0              : in std_logic;
            GND0              : in std_logic;
            mult_in1_opto: in std_logic_vector(31 downto 0);
            mult_in2_opto: in std_logic_vector(11 downto 0);
            add_in_opto: in std_logic_vector(75 downto 0);
            clk_opto    : in std_logic;
            clk_elec    : in std_logic;
            reset_elec  : in std_logic;
            add_sel_elec: in std_logic;
            mult_in1_elec: in std_logic_vector(4 downto 0);
            mult_in2_elec: in std_logic_vector(1 downto 0);
            add_in_elec: in std_logic_vector(4 downto 0);
            oe_sel_elec: in std_logic;
            power_short_1: out std_logic_vector(31 downto 0);
            power_short_2: out std_logic_vector(11 downto 0);
```

```
                power_short_3: out std_logic_vector(75 downto 0);
                power_short_4: out std_logic;
                power_short_5: out std_logic_vector(75 downto 0);
                data_out_opto: out std_logic_vector(75 downto 0);
                data_out_elec: out std_logic_vector(15 downto 0);
                dummy                    : out std_logic_vector(5 downto 0)
   );

attribute PACKAGE_NAME: string;
attribute PACKAGE_NAME of mult_32x12_76_chip: entity is “uncc442”;
end mult_32x12_76_chip;



architecture rgr_structural of mult_32x12_76_chip is

attribute PINNUM: integer;
attribute PAD: string;

component areapadin
port(
                PADPIN    : in std_logic;
                padtocore  : out std_logic
   );
end component;

component areapadout
port(
                coretopad  : in std_logic;
                PADPIN    : out std_logic
   );
end component;

component ap_vdd_20
port(
                PADPIN    : in std_logic
   );
end component;

component ap_gnd_20
port(
                PADPIN    : in std_logic
   );
end component;

component ap_wire
```

```
port(
        coretopad  : in std_logic;
        PADPIN     : out std_logic
   );
end component;

component mult_32x12_76
port(
        a          : in std_logic_vector(31 downto 0);
        b          : in std_logic_vector(11 downto 0);
        c          : in std_logic_vector(75 downto 0);
        d          : out std_logic_vector(75 downto 0)
   );
end component;


signal mult_in1_opto_int  : std_logic_vector(31 downto 0);
signal mult_in2_opto_int  : std_logic_vector(11 downto 0);
signal add_in_opto_int    : std_logic_vector(75 downto 0);
signal clk_opto_int       : std_logic;
signal data_out_opto_int: std_logic_vector(75 downto 0);
signal power_short_1_int: std_logic_vector(31 downto 0);
signal power_short_2_int: std_logic_vector(11 downto 0);
signal power_short_3_int: std_logic_vector(75 downto 0);
signal power_short_4_int: std_logic;
signal power_short_5_int: std_logic_vector(75 downto 0);
signal dummy_int          : std_logic_vector(5 downto 0);
signal mult_in1_opto_int_buff: std_logic_vector(31 downto 0);
signal mult_in2_opto_int_buff: std_logic_vector(11 downto 0);
signal add_in_opto_int_buff: std_logic_vector(75 downto 0);
signal clk_opto_int_buff: std_logic;
signal data_out_opto_int_buff: std_logic_vector(75 downto 0);
signal GND_27                       : std_logic_vector(26 downto 0);
signal GND_10                       : std_logic_vector(9 downto 0);
signal GND_71                       : std_logic_vector(70 downto 0);
signal data_out_elec_int: std_logic_vector(15 downto 0);
signal data_out_elec_int_buff: std_logic_vector(15 downto 0);
signal reset_elec_int     : std_logic;
signal reset_elec_int_buff: std_logic;
signal add_sel_elec_int   : std_logic;
signal add_sel_elec_int_buff: std_logic;
signal oe_sel_elec_int    : std_logic;
signal oe_sel_elec_int_buff: std_logic;
signal clk_elec_int       : std_logic;
signal clk_elec_int_buff: std_logic;
signal clk_in                       : std_logic;
```

```
signal mult_in1_elec_int: std_logic_vector(4 downto 0);
signal mult_in2_elec_int: std_logic_vector(1 downto 0);
signal mult_in1_elec_int_buff: std_logic_vector(4 downto 0);
signal mult_in2_elec_int_buff: std_logic_vector(1 downto 0);
signal add_in_elec_int      : std_logic_vector(4 downto 0);
signal add_in_elec_int_buff: std_logic_vector(4 downto 0);
signal mult_in1_high        : std_logic_vector(26 downto 0);
signal mult_in1_low         : std_logic_vector(4 downto 0);
signal mult_in2_high        : std_logic_vector(9 downto 0);
signal mult_in2_low         : std_logic_vector(1 downto 0);
signal add_in_high          : std_logic_vector(70 downto 0);
signal add_in_low           : std_logic_vector(4 downto 0);
signal add_in2                          : std_logic_vector(75 downto 0);
signal ff_in                            : std_logic_vector(75 downto 0);

attribute PINNUM of VDD_PAD0: label is 441;
attribute PINNUM of GND_PAD0: label is 442;

attribute PAD of VDD_PAD0: label is “VDD”;
attribute PAD of GND_PAD0: label is “GND”;

attribute PINNUM of IN_PAD0: label is 1;
attribute PINNUM of IN_PAD1: label is 2;
attribute PINNUM of IN_PAD2: label is 3;
attribute PINNUM of IN_PAD3: label is 4;
attribute PINNUM of IN_PAD4: label is 5;
attribute PINNUM of IN_PAD5: label is 6;
attribute PINNUM of IN_PAD6: label is 7;
attribute PINNUM of IN_PAD7: label is 8;
attribute PINNUM of IN_PAD8: label is 9;
attribute PINNUM of IN_PAD9: label is 10;
attribute PINNUM of IN_PAD10: label is 11;
attribute PINNUM of IN_PAD11: label is 12;
attribute PINNUM of IN_PAD12: label is 13;
attribute PINNUM of IN_PAD13: label is 14;
attribute PINNUM of IN_PAD14: label is 15;
attribute PINNUM of IN_PAD15: label is 16;
attribute PINNUM of IN_PAD16: label is 17;
attribute PINNUM of IN_PAD17: label is 18;
attribute PINNUM of IN_PAD18: label is 19;
attribute PINNUM of IN_PAD19: label is 20;
attribute PINNUM of IN_PAD20: label is 41;
attribute PINNUM of IN_PAD21: label is 42;
attribute PINNUM of IN_PAD22: label is 43;
attribute PINNUM of IN_PAD23: label is 44;
attribute PINNUM of IN_PAD24: label is 45;
```

```
attribute PINNUM of IN_PAD25: label is 46;
attribute PINNUM of IN_PAD26: label is 47;
attribute PINNUM of IN_PAD27: label is 48;
attribute PINNUM of IN_PAD28: label is 49;
attribute PINNUM of IN_PAD29: label is 50;
attribute PINNUM of IN_PAD30: label is 51;
attribute PINNUM of IN_PAD31: label is 52;
attribute PINNUM of IN_PAD32: label is 53;
attribute PINNUM of IN_PAD33: label is 54;
attribute PINNUM of IN_PAD34: label is 55;
attribute PINNUM of IN_PAD35: label is 56;
attribute PINNUM of IN_PAD36: label is 57;
attribute PINNUM of IN_PAD37: label is 58;
attribute PINNUM of IN_PAD38: label is 59;
attribute PINNUM of IN_PAD39: label is 60;
attribute PINNUM of IN_PAD40: label is 81;
attribute PINNUM of IN_PAD41: label is 82;
attribute PINNUM of IN_PAD42: label is 83;
attribute PINNUM of IN_PAD43: label is 84;
attribute PINNUM of IN_PAD44: label is 85;
attribute PINNUM of IN_PAD45: label is 86;
attribute PINNUM of IN_PAD46: label is 87;
attribute PINNUM of IN_PAD47: label is 88;
attribute PINNUM of IN_PAD48: label is 89;
attribute PINNUM of IN_PAD49: label is 90;
attribute PINNUM of IN_PAD50: label is 91;
attribute PINNUM of IN_PAD51: label is 92;
attribute PINNUM of IN_PAD52: label is 93;
attribute PINNUM of IN_PAD53: label is 94;
attribute PINNUM of IN_PAD54: label is 95;
attribute PINNUM of IN_PAD55: label is 96;
attribute PINNUM of IN_PAD56: label is 97;
attribute PINNUM of IN_PAD57: label is 98;
attribute PINNUM of IN_PAD58: label is 99;
attribute PINNUM of IN_PAD59: label is 100;
attribute PINNUM of IN_PAD60: label is 121;
attribute PINNUM of IN_PAD61: label is 122;
attribute PINNUM of IN_PAD62: label is 123;
attribute PINNUM of IN_PAD63: label is 124;
attribute PINNUM of IN_PAD64: label is 125;
attribute PINNUM of IN_PAD65: label is 126;
attribute PINNUM of IN_PAD66: label is 127;
attribute PINNUM of IN_PAD67: label is 128;
attribute PINNUM of IN_PAD68: label is 129;
attribute PINNUM of IN_PAD69: label is 130;
attribute PINNUM of IN_PAD70: label is 131;
```

```
attribute PINNUM of IN_PAD71: label is 132;
attribute PINNUM of IN_PAD72: label is 133;
attribute PINNUM of IN_PAD73: label is 134;
attribute PINNUM of IN_PAD74: label is 135;
attribute PINNUM of IN_PAD75: label is 136;
attribute PINNUM of IN_PAD76: label is 137;
attribute PINNUM of IN_PAD77: label is 138;
attribute PINNUM of IN_PAD78: label is 139;
attribute PINNUM of IN_PAD79: label is 140;
attribute PINNUM of IN_PAD80: label is 161;
attribute PINNUM of IN_PAD81: label is 162;
attribute PINNUM of IN_PAD82: label is 163;
attribute PINNUM of IN_PAD83: label is 164;
attribute PINNUM of IN_PAD84: label is 165;
attribute PINNUM of IN_PAD85: label is 166;
attribute PINNUM of IN_PAD86: label is 167;
attribute PINNUM of IN_PAD87: label is 168;
attribute PINNUM of IN_PAD88: label is 169;
attribute PINNUM of IN_PAD89: label is 170;
attribute PINNUM of IN_PAD90: label is 171;
attribute PINNUM of IN_PAD91: label is 172;
attribute PINNUM of IN_PAD92: label is 173;
attribute PINNUM of IN_PAD93: label is 174;
attribute PINNUM of IN_PAD94: label is 175;
attribute PINNUM of IN_PAD95: label is 176;
attribute PINNUM of IN_PAD96: label is 177;
attribute PINNUM of IN_PAD97: label is 178;
attribute PINNUM of IN_PAD98: label is 179;
attribute PINNUM of IN_PAD99: label is 180;
attribute PINNUM of IN_PAD100: label is 201;
attribute PINNUM of IN_PAD101: label is 202;
attribute PINNUM of IN_PAD102: label is 203;
attribute PINNUM of IN_PAD103: label is 204;
attribute PINNUM of IN_PAD104: label is 205;
attribute PINNUM of IN_PAD105: label is 206;
attribute PINNUM of IN_PAD106: label is 207;
attribute PINNUM of IN_PAD107: label is 208;
attribute PINNUM of IN_PAD108: label is 209;
attribute PINNUM of IN_PAD109: label is 210;
attribute PINNUM of IN_PAD110: label is 211;
attribute PINNUM of IN_PAD111: label is 212;
attribute PINNUM of IN_PAD112: label is 213;
attribute PINNUM of IN_PAD113: label is 214;
attribute PINNUM of IN_PAD114: label is 215;
attribute PINNUM of IN_PAD115: label is 216;
attribute PINNUM of IN_PAD116: label is 217;
```

```
attribute PINNUM of IN_PAD117: label is 218;
attribute PINNUM of IN_PAD118: label is 219;
attribute PINNUM of IN_PAD119: label is 220;
attribute PINNUM of IN_PAD120: label is 241;
attribute PINNUM of IN_PAD121: label is 411;
attribute PINNUM of IN_PAD122: label is 412;
attribute PINNUM of IN_PAD123: label is 413;
attribute PINNUM of IN_PAD124: label is 415;
attribute PINNUM of IN_PAD125: label is 416;
attribute PINNUM of IN_PAD126: label is 418;
attribute PINNUM of IN_PAD127: label is 419;
attribute PINNUM of IN_PAD128: label is 420;
attribute PINNUM of IN_PAD129: label is 431;
attribute PINNUM of IN_PAD130: label is 432;
attribute PINNUM of IN_PAD131: label is 433;
attribute PINNUM of IN_PAD132: label is 435;
attribute PINNUM of IN_PAD133: label is 436;
attribute PINNUM of IN_PAD134: label is 438;
attribute PINNUM of IN_PAD135: label is 439;
attribute PINNUM of IN_PAD136: label is 440;

attribute PAD of IN_PAD0: label is “PAD”;
attribute PAD of IN_PAD1: label is “PAD”;
attribute PAD of IN_PAD2: label is “PAD”;
attribute PAD of IN_PAD3: label is “PAD”;
attribute PAD of IN_PAD4: label is “PAD”;
attribute PAD of IN_PAD5: label is “PAD”;
attribute PAD of IN_PAD6: label is “PAD”;
attribute PAD of IN_PAD7: label is “PAD”;
attribute PAD of IN_PAD8: label is “PAD”;
attribute PAD of IN_PAD9: label is “PAD”;
attribute PAD of IN_PAD10: label is “PAD”;
attribute PAD of IN_PAD11: label is “PAD”;
attribute PAD of IN_PAD12: label is “PAD”;
attribute PAD of IN_PAD13: label is “PAD”;
attribute PAD of IN_PAD14: label is “PAD”;
attribute PAD of IN_PAD15: label is “PAD”;
attribute PAD of IN_PAD16: label is “PAD”;
attribute PAD of IN_PAD17: label is “PAD”;
attribute PAD of IN_PAD18: label is “PAD”;
attribute PAD of IN_PAD19: label is “PAD”;
attribute PAD of IN_PAD20: label is “PAD”;
attribute PAD of IN_PAD21: label is “PAD”;
attribute PAD of IN_PAD22: label is “PAD”;
attribute PAD of IN_PAD23: label is “PAD”;
attribute PAD of IN_PAD24: label is “PAD”;
```

attribute PAD of IN_PAD25: label is “PAD”;
attribute PAD of IN_PAD26: label is “PAD”;
attribute PAD of IN_PAD27: label is “PAD”;
attribute PAD of IN_PAD28: label is “PAD”;
attribute PAD of IN_PAD29: label is “PAD”;
attribute PAD of IN_PAD30: label is “PAD”;
attribute PAD of IN_PAD31: label is “PAD”;
attribute PAD of IN_PAD32: label is “PAD”;
attribute PAD of IN_PAD33: label is “PAD”;
attribute PAD of IN_PAD34: label is “PAD”;
attribute PAD of IN_PAD35: label is “PAD”;
attribute PAD of IN_PAD36: label is “PAD”;
attribute PAD of IN_PAD37: label is “PAD”;
attribute PAD of IN_PAD38: label is “PAD”;
attribute PAD of IN_PAD39: label is “PAD”;
attribute PAD of IN_PAD40: label is “PAD”;
attribute PAD of IN_PAD41: label is “PAD”;
attribute PAD of IN_PAD42: label is “PAD”;
attribute PAD of IN_PAD43: label is “PAD”;
attribute PAD of IN_PAD44: label is “PAD”;
attribute PAD of IN_PAD45: label is “PAD”;
attribute PAD of IN_PAD46: label is “PAD”;
attribute PAD of IN_PAD47: label is “PAD”;
attribute PAD of IN_PAD48: label is “PAD”;
attribute PAD of IN_PAD49: label is “PAD”;
attribute PAD of IN_PAD50: label is “PAD”;
attribute PAD of IN_PAD51: label is “PAD”;
attribute PAD of IN_PAD52: label is “PAD”;
attribute PAD of IN_PAD53: label is “PAD”;
attribute PAD of IN_PAD54: label is “PAD”;
attribute PAD of IN_PAD55: label is “PAD”;
attribute PAD of IN_PAD56: label is “PAD”;
attribute PAD of IN_PAD57: label is “PAD”;
attribute PAD of IN_PAD58: label is “PAD”;
attribute PAD of IN_PAD59: label is “PAD”;
attribute PAD of IN_PAD60: label is “PAD”;
attribute PAD of IN_PAD61: label is “PAD”;
attribute PAD of IN_PAD62: label is “PAD”;
attribute PAD of IN_PAD63: label is “PAD”;
attribute PAD of IN_PAD64: label is “PAD”;
attribute PAD of IN_PAD65: label is “PAD”;
attribute PAD of IN_PAD66: label is “PAD”;
attribute PAD of IN_PAD67: label is “PAD”;
attribute PAD of IN_PAD68: label is “PAD”;
attribute PAD of IN_PAD69: label is “PAD”;
attribute PAD of IN_PAD70: label is “PAD”;

attribute PAD of IN_PAD71: label is “PAD”;
attribute PAD of IN_PAD72: label is “PAD”;
attribute PAD of IN_PAD73: label is “PAD”;
attribute PAD of IN_PAD74: label is “PAD”;
attribute PAD of IN_PAD75: label is “PAD”;
attribute PAD of IN_PAD76: label is “PAD”;
attribute PAD of IN_PAD77: label is “PAD”;
attribute PAD of IN_PAD78: label is “PAD”;
attribute PAD of IN_PAD79: label is “PAD”;
attribute PAD of IN_PAD80: label is “PAD”;
attribute PAD of IN_PAD81: label is “PAD”;
attribute PAD of IN_PAD82: label is “PAD”;
attribute PAD of IN_PAD83: label is “PAD”;
attribute PAD of IN_PAD84: label is “PAD”;
attribute PAD of IN_PAD85: label is “PAD”;
attribute PAD of IN_PAD86: label is “PAD”;
attribute PAD of IN_PAD87: label is “PAD”;
attribute PAD of IN_PAD88: label is “PAD”;
attribute PAD of IN_PAD89: label is “PAD”;
attribute PAD of IN_PAD90: label is “PAD”;
attribute PAD of IN_PAD91: label is “PAD”;
attribute PAD of IN_PAD92: label is “PAD”;
attribute PAD of IN_PAD93: label is “PAD”;
attribute PAD of IN_PAD94: label is “PAD”;
attribute PAD of IN_PAD95: label is “PAD”;
attribute PAD of IN_PAD96: label is “PAD”;
attribute PAD of IN_PAD97: label is “PAD”;
attribute PAD of IN_PAD98: label is “PAD”;
attribute PAD of IN_PAD99: label is “PAD”;
attribute PAD of IN_PAD100: label is “PAD”;
attribute PAD of IN_PAD101: label is “PAD”;
attribute PAD of IN_PAD102: label is “PAD”;
attribute PAD of IN_PAD103: label is “PAD”;
attribute PAD of IN_PAD104: label is “PAD”;
attribute PAD of IN_PAD105: label is “PAD”;
attribute PAD of IN_PAD106: label is “PAD”;
attribute PAD of IN_PAD107: label is “PAD”;
attribute PAD of IN_PAD108: label is “PAD”;
attribute PAD of IN_PAD109: label is “PAD”;
attribute PAD of IN_PAD110: label is “PAD”;
attribute PAD of IN_PAD111: label is “PAD”;
attribute PAD of IN_PAD112: label is “PAD”;
attribute PAD of IN_PAD113: label is “PAD”;
attribute PAD of IN_PAD114: label is “PAD”;
attribute PAD of IN_PAD115: label is “PAD”;
attribute PAD of IN_PAD116: label is “PAD”;

```
attribute PAD of IN_PAD117: label is “PAD”;
attribute PAD of IN_PAD118: label is “PAD”;
attribute PAD of IN_PAD119: label is “PAD”;
attribute PAD of IN_PAD120: label is “PAD”;
attribute PAD of IN_PAD121: label is “PAD”;
attribute PAD of IN_PAD122: label is “PAD”;
attribute PAD of IN_PAD123: label is “PAD”;
attribute PAD of IN_PAD124: label is “PAD”;
attribute PAD of IN_PAD125: label is “PAD”;
attribute PAD of IN_PAD126: label is “PAD”;
attribute PAD of IN_PAD127: label is “PAD”;
attribute PAD of IN_PAD128: label is “PAD”;
attribute PAD of IN_PAD129: label is “PAD”;
attribute PAD of IN_PAD130: label is “PAD”;
attribute PAD of IN_PAD131: label is “PAD”;
attribute PAD of IN_PAD132: label is “PAD”;
attribute PAD of IN_PAD133: label is “PAD”;
attribute PAD of IN_PAD134: label is “PAD”;
attribute PAD of IN_PAD135: label is “PAD”;
attribute PAD of IN_PAD136: label is “PAD”;

attribute PINNUM of OUT_PAD0: label is 242;
attribute PINNUM of OUT_PAD1: label is 243;
attribute PINNUM of OUT_PAD2: label is 244;
attribute PINNUM of OUT_PAD3: label is 245;
attribute PINNUM of OUT_PAD4: label is 246;
attribute PINNUM of OUT_PAD5: label is 247;
attribute PINNUM of OUT_PAD6: label is 248;
attribute PINNUM of OUT_PAD7: label is 249;
attribute PINNUM of OUT_PAD8: label is 250;
attribute PINNUM of OUT_PAD9: label is 251;
attribute PINNUM of OUT_PAD10: label is 252;
attribute PINNUM of OUT_PAD11: label is 253;
attribute PINNUM of OUT_PAD12: label is 254;
attribute PINNUM of OUT_PAD13: label is 255;
attribute PINNUM of OUT_PAD14: label is 256;
attribute PINNUM of OUT_PAD15: label is 257;
attribute PINNUM of OUT_PAD16: label is 258;
attribute PINNUM of OUT_PAD17: label is 259;
attribute PINNUM of OUT_PAD18: label is 260;
attribute PINNUM of OUT_PAD19: label is 281;
attribute PINNUM of OUT_PAD20: label is 282;
attribute PINNUM of OUT_PAD21: label is 283;
attribute PINNUM of OUT_PAD22: label is 284;
attribute PINNUM of OUT_PAD23: label is 285;
attribute PINNUM of OUT_PAD24: label is 286;
```

```
attribute PINNUM of OUT_PAD25: label is 287;
attribute PINNUM of OUT_PAD26: label is 288;
attribute PINNUM of OUT_PAD27: label is 289;
attribute PINNUM of OUT_PAD28: label is 290;
attribute PINNUM of OUT_PAD29: label is 291;
attribute PINNUM of OUT_PAD30: label is 292;
attribute PINNUM of OUT_PAD31: label is 293;
attribute PINNUM of OUT_PAD32: label is 294;
attribute PINNUM of OUT_PAD33: label is 295;
attribute PINNUM of OUT_PAD34: label is 296;
attribute PINNUM of OUT_PAD35: label is 297;
attribute PINNUM of OUT_PAD36: label is 298;
attribute PINNUM of OUT_PAD37: label is 299;
attribute PINNUM of OUT_PAD38: label is 300;
attribute PINNUM of OUT_PAD39: label is 321;
attribute PINNUM of OUT_PAD40: label is 322;
attribute PINNUM of OUT_PAD41: label is 323;
attribute PINNUM of OUT_PAD42: label is 324;
attribute PINNUM of OUT_PAD43: label is 325;
attribute PINNUM of OUT_PAD44: label is 326;
attribute PINNUM of OUT_PAD45: label is 327;
attribute PINNUM of OUT_PAD46: label is 328;
attribute PINNUM of OUT_PAD47: label is 329;
attribute PINNUM of OUT_PAD48: label is 330;
attribute PINNUM of OUT_PAD49: label is 331;
attribute PINNUM of OUT_PAD50: label is 332;
attribute PINNUM of OUT_PAD51: label is 333;
attribute PINNUM of OUT_PAD52: label is 334;
attribute PINNUM of OUT_PAD53: label is 335;
attribute PINNUM of OUT_PAD54: label is 336;
attribute PINNUM of OUT_PAD55: label is 337;
attribute PINNUM of OUT_PAD56: label is 338;
attribute PINNUM of OUT_PAD57: label is 339;
attribute PINNUM of OUT_PAD58: label is 340;
attribute PINNUM of OUT_PAD59: label is 361;
attribute PINNUM of OUT_PAD60: label is 362;
attribute PINNUM of OUT_PAD61: label is 363;
attribute PINNUM of OUT_PAD62: label is 364;
attribute PINNUM of OUT_PAD63: label is 365;
attribute PINNUM of OUT_PAD64: label is 366;
attribute PINNUM of OUT_PAD65: label is 367;
attribute PINNUM of OUT_PAD66: label is 368;
attribute PINNUM of OUT_PAD67: label is 369;
attribute PINNUM of OUT_PAD68: label is 370;
attribute PINNUM of OUT_PAD69: label is 371;
attribute PINNUM of OUT_PAD70: label is 372;
```

attribute PINNUM of OUT_PAD71: label is 373;
attribute PINNUM of OUT_PAD72: label is 374;
attribute PINNUM of OUT_PAD73: label is 375;
attribute PINNUM of OUT_PAD74: label is 376;
attribute PINNUM of OUT_PAD75: label is 377;

attribute PAD of OUT_PAD0: label is “PAD”;
attribute PAD of OUT_PAD1: label is “PAD”;
attribute PAD of OUT_PAD2: label is “PAD”;
attribute PAD of OUT_PAD3: label is “PAD”;
attribute PAD of OUT_PAD4: label is “PAD”;
attribute PAD of OUT_PAD5: label is “PAD”;
attribute PAD of OUT_PAD6: label is “PAD”;
attribute PAD of OUT_PAD7: label is “PAD”;
attribute PAD of OUT_PAD8: label is “PAD”;
attribute PAD of OUT_PAD9: label is “PAD”;
attribute PAD of OUT_PAD10: label is “PAD”;
attribute PAD of OUT_PAD11: label is “PAD”;
attribute PAD of OUT_PAD12: label is “PAD”;
attribute PAD of OUT_PAD13: label is “PAD”;
attribute PAD of OUT_PAD14: label is “PAD”;
attribute PAD of OUT_PAD15: label is “PAD”;
attribute PAD of OUT_PAD16: label is “PAD”;
attribute PAD of OUT_PAD17: label is “PAD”;
attribute PAD of OUT_PAD18: label is “PAD”;
attribute PAD of OUT_PAD19: label is “PAD”;
attribute PAD of OUT_PAD20: label is “PAD”;
attribute PAD of OUT_PAD21: label is “PAD”;
attribute PAD of OUT_PAD22: label is “PAD”;
attribute PAD of OUT_PAD23: label is “PAD”;
attribute PAD of OUT_PAD24: label is “PAD”;
attribute PAD of OUT_PAD25: label is “PAD”;
attribute PAD of OUT_PAD26: label is “PAD”;
attribute PAD of OUT_PAD27: label is “PAD”;
attribute PAD of OUT_PAD28: label is “PAD”;
attribute PAD of OUT_PAD29: label is “PAD”;
attribute PAD of OUT_PAD30: label is “PAD”;
attribute PAD of OUT_PAD31: label is “PAD”;
attribute PAD of OUT_PAD32: label is “PAD”;
attribute PAD of OUT_PAD33: label is “PAD”;
attribute PAD of OUT_PAD34: label is “PAD”;
attribute PAD of OUT_PAD35: label is “PAD”;
attribute PAD of OUT_PAD36: label is “PAD”;
attribute PAD of OUT_PAD37: label is “PAD”;
attribute PAD of OUT_PAD38: label is “PAD”;
attribute PAD of OUT_PAD39: label is “PAD”;

attribute PAD of OUT_PAD40: label is “PAD”;
attribute PAD of OUT_PAD41: label is “PAD”;
attribute PAD of OUT_PAD42: label is “PAD”;
attribute PAD of OUT_PAD43: label is “PAD”;
attribute PAD of OUT_PAD44: label is “PAD”;
attribute PAD of OUT_PAD45: label is “PAD”;
attribute PAD of OUT_PAD46: label is “PAD”;
attribute PAD of OUT_PAD47: label is “PAD”;
attribute PAD of OUT_PAD48: label is “PAD”;
attribute PAD of OUT_PAD49: label is “PAD”;
attribute PAD of OUT_PAD50: label is “PAD”;
attribute PAD of OUT_PAD51: label is “PAD”;
attribute PAD of OUT_PAD52: label is “PAD”;
attribute PAD of OUT_PAD53: label is “PAD”;
attribute PAD of OUT_PAD54: label is “PAD”;
attribute PAD of OUT_PAD55: label is “PAD”;
attribute PAD of OUT_PAD56: label is “PAD”;
attribute PAD of OUT_PAD57: label is “PAD”;
attribute PAD of OUT_PAD58: label is “PAD”;
attribute PAD of OUT_PAD59: label is “PAD”;
attribute PAD of OUT_PAD60: label is “PAD”;
attribute PAD of OUT_PAD61: label is “PAD”;
attribute PAD of OUT_PAD62: label is “PAD”;
attribute PAD of OUT_PAD63: label is “PAD”;
attribute PAD of OUT_PAD64: label is “PAD”;
attribute PAD of OUT_PAD65: label is “PAD”;
attribute PAD of OUT_PAD66: label is “PAD”;
attribute PAD of OUT_PAD67: label is “PAD”;
attribute PAD of OUT_PAD68: label is “PAD”;
attribute PAD of OUT_PAD69: label is “PAD”;
attribute PAD of OUT_PAD70: label is “PAD”;
attribute PAD of OUT_PAD71: label is “PAD”;
attribute PAD of OUT_PAD72: label is “PAD”;
attribute PAD of OUT_PAD73: label is “PAD”;
attribute PAD of OUT_PAD74: label is “PAD”;
attribute PAD of OUT_PAD75: label is “PAD”;
attribute PAD of OUT_PAD76: label is “PAD”;
attribute PAD of OUT_PAD77: label is “PAD”;
attribute PAD of OUT_PAD78: label is “PAD”;
attribute PAD of OUT_PAD79: label is “PAD”;
attribute PAD of OUT_PAD80: label is “PAD”;
attribute PAD of OUT_PAD81: label is “PAD”;
attribute PAD of OUT_PAD82: label is “PAD”;
attribute PAD of OUT_PAD83: label is “PAD”;
attribute PAD of OUT_PAD84: label is “PAD”;
attribute PAD of OUT_PAD85: label is “PAD”;

attribute PAD of OUT_PAD86: label is “PAD”;
attribute PAD of OUT_PAD87: label is “PAD”;
attribute PAD of OUT_PAD88: label is “PAD”;
attribute PAD of OUT_PAD89: label is “PAD”;
attribute PAD of OUT_PAD90: label is “PAD”;
attribute PAD of OUT_PAD91: label is “PAD”;
attribute PAD of OUT_PAD92: label is “PAD”;
attribute PAD of OUT_PAD93: label is “PAD”;
attribute PAD of OUT_PAD94: label is “PAD”;
attribute PAD of OUT_PAD95: label is “PAD”;
attribute PAD of OUT_PAD96: label is “PAD”;
attribute PAD of OUT_PAD97: label is “PAD”;
attribute PAD of OUT_PAD98: label is “PAD”;
attribute PAD of OUT_PAD99: label is “PAD”;
attribute PAD of OUT_PAD100: label is “PAD”;
attribute PAD of OUT_PAD101: label is “PAD”;
attribute PAD of OUT_PAD102: label is “PAD”;
attribute PAD of OUT_PAD103: label is “PAD”;
attribute PAD of OUT_PAD104: label is “PAD”;
attribute PAD of OUT_PAD105: label is “PAD”;
attribute PAD of OUT_PAD106: label is “PAD”;
attribute PAD of OUT_PAD107: label is “PAD”;
attribute PAD of OUT_PAD108: label is “PAD”;
attribute PAD of OUT_PAD109: label is “PAD”;
attribute PAD of OUT_PAD110: label is “PAD”;
attribute PAD of OUT_PAD111: label is “PAD”;
attribute PAD of OUT_PAD112: label is “PAD”;
attribute PAD of OUT_PAD113: label is “PAD”;
attribute PAD of OUT_PAD114: label is “PAD”;
attribute PAD of OUT_PAD115: label is “PAD”;
attribute PAD of OUT_PAD116: label is “PAD”;
attribute PAD of OUT_PAD117: label is “PAD”;
attribute PAD of OUT_PAD118: label is “PAD”;
attribute PAD of OUT_PAD119: label is “PAD”;
attribute PAD of OUT_PAD120: label is “PAD”;
attribute PAD of OUT_PAD121: label is “PAD”;
attribute PAD of OUT_PAD122: label is “PAD”;
attribute PAD of OUT_PAD123: label is “PAD”;
attribute PAD of OUT_PAD124: label is “PAD”;
attribute PAD of OUT_PAD125: label is “PAD”;
attribute PAD of OUT_PAD126: label is “PAD”;
attribute PAD of OUT_PAD127: label is “PAD”;
attribute PAD of OUT_PAD128: label is “PAD”;
attribute PAD of OUT_PAD129: label is “PAD”;
attribute PAD of OUT_PAD130: label is “PAD”;
attribute PAD of OUT_PAD131: label is “PAD”;

attribute PAD of OUT_PAD132: label is “PAD”;
attribute PAD of OUT_PAD133: label is “PAD”;
attribute PAD of OUT_PAD134: label is “PAD”;
attribute PAD of OUT_PAD135: label is “PAD”;
attribute PAD of OUT_PAD136: label is “PAD”;
attribute PAD of OUT_PAD137: label is “PAD”;
attribute PAD of OUT_PAD138: label is “PAD”;
attribute PAD of OUT_PAD139: label is “PAD”;
attribute PAD of OUT_PAD140: label is “PAD”;
attribute PAD of OUT_PAD141: label is “PAD”;
attribute PAD of OUT_PAD142: label is “PAD”;
attribute PAD of OUT_PAD143: label is “PAD”;
attribute PAD of OUT_PAD144: label is “PAD”;
attribute PAD of OUT_PAD145: label is “PAD”;
attribute PAD of OUT_PAD146: label is “PAD”;
attribute PAD of OUT_PAD147: label is “PAD”;
attribute PAD of OUT_PAD148: label is “PAD”;
attribute PAD of OUT_PAD149: label is “PAD”;
attribute PAD of OUT_PAD150: label is “PAD”;
attribute PAD of OUT_PAD151: label is “PAD”;
attribute PAD of OUT_PAD152: label is “PAD”;
attribute PAD of OUT_PAD153: label is “PAD”;
attribute PAD of OUT_PAD154: label is “PAD”;
attribute PAD of OUT_PAD155: label is “PAD”;
attribute PAD of OUT_PAD156: label is “PAD”;
attribute PAD of OUT_PAD157: label is “PAD”;
attribute PAD of OUT_PAD158: label is “PAD”;
attribute PAD of OUT_PAD159: label is “PAD”;
attribute PAD of OUT_PAD160: label is “PAD”;
attribute PAD of OUT_PAD161: label is “PAD”;
attribute PAD of OUT_PAD162: label is “PAD”;
attribute PAD of OUT_PAD163: label is “PAD”;
attribute PAD of OUT_PAD164: label is “PAD”;
attribute PAD of OUT_PAD165: label is “PAD”;
attribute PAD of OUT_PAD166: label is “PAD”;
attribute PAD of OUT_PAD167: label is “PAD”;
attribute PAD of OUT_PAD168: label is “PAD”;
attribute PAD of OUT_PAD169: label is “PAD”;
attribute PAD of OUT_PAD170: label is “PAD”;
attribute PAD of OUT_PAD171: label is “PAD”;
attribute PAD of OUT_PAD172: label is “PAD”;
attribute PAD of OUT_PAD173: label is “PAD”;
attribute PAD of OUT_PAD174: label is “PAD”;
attribute PAD of OUT_PAD175: label is “PAD”;
attribute PAD of OUT_PAD176: label is “PAD”;
attribute PAD of OUT_PAD177: label is “PAD”;

attribute PAD of OUT_PAD178: label is “PAD”;
attribute PAD of OUT_PAD179: label is “PAD”;
attribute PAD of OUT_PAD180: label is “PAD”;
attribute PAD of OUT_PAD181: label is “PAD”;
attribute PAD of OUT_PAD182: label is “PAD”;
attribute PAD of OUT_PAD183: label is “PAD”;
attribute PAD of OUT_PAD184: label is “PAD”;
attribute PAD of OUT_PAD185: label is “PAD”;
attribute PAD of OUT_PAD186: label is “PAD”;
attribute PAD of OUT_PAD187: label is “PAD”;
attribute PAD of OUT_PAD188: label is “PAD”;
attribute PAD of OUT_PAD189: label is “PAD”;
attribute PAD of OUT_PAD190: label is “PAD”;
attribute PAD of OUT_PAD191: label is “PAD”;
attribute PAD of OUT_PAD192: label is “PAD”;
attribute PAD of OUT_PAD193: label is “PAD”;
attribute PAD of OUT_PAD194: label is “PAD”;
attribute PAD of OUT_PAD195: label is “PAD”;
attribute PAD of OUT_PAD196: label is “PAD”;
attribute PAD of OUT_PAD197: label is “PAD”;
attribute PAD of OUT_PAD198: label is “PAD”;
attribute PAD of OUT_PAD199: label is “PAD”;
attribute PAD of OUT_PAD200: label is “PAD”;
attribute PAD of OUT_PAD201: label is “PAD”;
attribute PAD of OUT_PAD202: label is “PAD”;
attribute PAD of OUT_PAD203: label is “PAD”;
attribute PAD of OUT_PAD204: label is “PAD”;
attribute PAD of OUT_PAD205: label is “PAD”;
attribute PAD of OUT_PAD206: label is “PAD”;
attribute PAD of OUT_PAD207: label is “PAD”;
attribute PAD of OUT_PAD208: label is “PAD”;
attribute PAD of OUT_PAD209: label is “PAD”;
attribute PAD of OUT_PAD210: label is “PAD”;
attribute PAD of OUT_PAD211: label is “PAD”;
attribute PAD of OUT_PAD212: label is “PAD”;
attribute PAD of OUT_PAD213: label is “PAD”;
attribute PAD of OUT_PAD214: label is “PAD”;
attribute PAD of OUT_PAD215: label is “PAD”;
attribute PAD of OUT_PAD216: label is “PAD”;
attribute PAD of OUT_PAD217: label is “PAD”;
attribute PAD of OUT_PAD218: label is “PAD”;
attribute PAD of OUT_PAD219: label is “PAD”;
attribute PAD of OUT_PAD220: label is “PAD”;
attribute PAD of OUT_PAD221: label is “PAD”;
attribute PAD of OUT_PAD222: label is “PAD”;
attribute PAD of OUT_PAD223: label is “PAD”;

attribute PAD of OUT_PAD224: label is “PAD”;
attribute PAD of OUT_PAD225: label is “PAD”;
attribute PAD of OUT_PAD226: label is “PAD”;
attribute PAD of OUT_PAD227: label is “PAD”;
attribute PAD of OUT_PAD228: label is “PAD”;
attribute PAD of OUT_PAD229: label is “PAD”;
attribute PAD of OUT_PAD230: label is “PAD”;
attribute PAD of OUT_PAD231: label is “PAD”;
attribute PAD of OUT_PAD232: label is “PAD”;
attribute PAD of OUT_PAD233: label is “PAD”;
attribute PAD of OUT_PAD234: label is “PAD”;
attribute PAD of OUT_PAD235: label is “PAD”;
attribute PAD of OUT_PAD236: label is “PAD”;
attribute PAD of OUT_PAD237: label is “PAD”;
attribute PAD of OUT_PAD238: label is “PAD”;
attribute PAD of OUT_PAD239: label is “PAD”;
attribute PAD of OUT_PAD240: label is “PAD”;
attribute PAD of OUT_PAD241: label is “PAD”;
attribute PAD of OUT_PAD242: label is “PAD”;
attribute PAD of OUT_PAD243: label is “PAD”;
attribute PAD of OUT_PAD244: label is “PAD”;
attribute PAD of OUT_PAD245: label is “PAD”;
attribute PAD of OUT_PAD246: label is “PAD”;
attribute PAD of OUT_PAD247: label is “PAD”;
attribute PAD of OUT_PAD248: label is “PAD”;
attribute PAD of OUT_PAD249: label is “PAD”;
attribute PAD of OUT_PAD250: label is “PAD”;
attribute PAD of OUT_PAD251: label is “PAD”;
attribute PAD of OUT_PAD252: label is “PAD”;
attribute PAD of OUT_PAD253: label is “PAD”;
attribute PAD of OUT_PAD254: label is “PAD”;
attribute PAD of OUT_PAD255: label is “PAD”;
attribute PAD of OUT_PAD256: label is “PAD”;
attribute PAD of OUT_PAD257: label is “PAD”;
attribute PAD of OUT_PAD258: label is “PAD”;
attribute PAD of OUT_PAD259: label is “PAD”;
attribute PAD of OUT_PAD260: label is “PAD”;
attribute PAD of OUT_PAD261: label is “PAD”;
attribute PAD of OUT_PAD262: label is “PAD”;
attribute PAD of OUT_PAD263: label is “PAD”;
attribute PAD of OUT_PAD264: label is “PAD”;
attribute PAD of OUT_PAD265: label is “PAD”;
attribute PAD of OUT_PAD266: label is “PAD”;
attribute PAD of OUT_PAD267: label is “PAD”;
attribute PAD of OUT_PAD268: label is “PAD”;
attribute PAD of OUT_PAD269: label is “PAD”;

attribute PAD of OUT_PAD270: label is “PAD”;
attribute PAD of OUT_PAD271: label is “PAD”;
attribute PAD of OUT_PAD272: label is “PAD”;
attribute PAD of OUT_PAD273: label is “PAD”;
attribute PAD of OUT_PAD274: label is “PAD”;
attribute PAD of OUT_PAD275: label is “PAD”;
attribute PAD of OUT_PAD276: label is “PAD”;
attribute PAD of OUT_PAD277: label is “PAD”;
attribute PAD of OUT_PAD278: label is “PAD”;
attribute PAD of OUT_PAD279: label is “PAD”;
attribute PAD of OUT_PAD280: label is “PAD”;
attribute PAD of OUT_PAD281: label is “PAD”;
attribute PAD of OUT_PAD282: label is “PAD”;
attribute PAD of OUT_PAD283: label is “PAD”;
attribute PAD of OUT_PAD284: label is “PAD”;
attribute PAD of OUT_PAD285: label is “PAD”;
attribute PAD of OUT_PAD286: label is “PAD”;
attribute PAD of OUT_PAD287: label is “PAD”;
attribute PAD of OUT_PAD288: label is “PAD”;
attribute PAD of OUT_PAD289: label is “PAD”;
attribute PAD of OUT_PAD290: label is “PAD”;
attribute PAD of OUT_PAD291: label is “PAD”;
attribute PAD of OUT_PAD292: label is “PAD”;
attribute PAD of OUT_PAD293: label is “PAD”;
attribute PAD of OUT_PAD294: label is “PAD”;

attribute PINNUM of OUT_PAD76: label is 21;
attribute PINNUM of OUT_PAD77: label is 22;
attribute PINNUM of OUT_PAD78: label is 23;
attribute PINNUM of OUT_PAD79: label is 24;
attribute PINNUM of OUT_PAD80: label is 25;
attribute PINNUM of OUT_PAD81: label is 26;
attribute PINNUM of OUT_PAD82: label is 27;
attribute PINNUM of OUT_PAD83: label is 28;
attribute PINNUM of OUT_PAD84: label is 29;
attribute PINNUM of OUT_PAD85: label is 30;
attribute PINNUM of OUT_PAD86: label is 31;
attribute PINNUM of OUT_PAD87: label is 32;
attribute PINNUM of OUT_PAD88: label is 33;
attribute PINNUM of OUT_PAD89: label is 34;
attribute PINNUM of OUT_PAD90: label is 35;
attribute PINNUM of OUT_PAD91: label is 36;
attribute PINNUM of OUT_PAD92: label is 37;
attribute PINNUM of OUT_PAD93: label is 38;
attribute PINNUM of OUT_PAD94: label is 39;
attribute PINNUM of OUT_PAD95: label is 40;

```
attribute PINNUM of OUT_PAD96: label is 61;
attribute PINNUM of OUT_PAD97: label is 62;
attribute PINNUM of OUT_PAD98: label is 63;
attribute PINNUM of OUT_PAD99: label is 64;
attribute PINNUM of OUT_PAD100: label is 65;
attribute PINNUM of OUT_PAD101: label is 66;
attribute PINNUM of OUT_PAD102: label is 67;
attribute PINNUM of OUT_PAD103: label is 68;
attribute PINNUM of OUT_PAD104: label is 69;
attribute PINNUM of OUT_PAD105: label is 70;
attribute PINNUM of OUT_PAD106: label is 71;
attribute PINNUM of OUT_PAD107: label is 72;
attribute PINNUM of OUT_PAD108: label is 73;
attribute PINNUM of OUT_PAD109: label is 74;
attribute PINNUM of OUT_PAD110: label is 75;
attribute PINNUM of OUT_PAD111: label is 76;
attribute PINNUM of OUT_PAD112: label is 77;
attribute PINNUM of OUT_PAD113: label is 78;
attribute PINNUM of OUT_PAD114: label is 79;
attribute PINNUM of OUT_PAD115: label is 80;
attribute PINNUM of OUT_PAD116: label is 101;
attribute PINNUM of OUT_PAD117: label is 102;
attribute PINNUM of OUT_PAD118: label is 103;
attribute PINNUM of OUT_PAD119: label is 104;
attribute PINNUM of OUT_PAD120: label is 105;
attribute PINNUM of OUT_PAD121: label is 106;
attribute PINNUM of OUT_PAD122: label is 107;
attribute PINNUM of OUT_PAD123: label is 108;
attribute PINNUM of OUT_PAD124: label is 109;
attribute PINNUM of OUT_PAD125: label is 110;
attribute PINNUM of OUT_PAD126: label is 111;
attribute PINNUM of OUT_PAD127: label is 112;
attribute PINNUM of OUT_PAD128: label is 113;
attribute PINNUM of OUT_PAD129: label is 114;
attribute PINNUM of OUT_PAD130: label is 115;
attribute PINNUM of OUT_PAD131: label is 116;
attribute PINNUM of OUT_PAD132: label is 117;
attribute PINNUM of OUT_PAD133: label is 118;
attribute PINNUM of OUT_PAD134: label is 119;
attribute PINNUM of OUT_PAD135: label is 120;
attribute PINNUM of OUT_PAD136: label is 141;
attribute PINNUM of OUT_PAD137: label is 142;
attribute PINNUM of OUT_PAD138: label is 143;
attribute PINNUM of OUT_PAD139: label is 144;
attribute PINNUM of OUT_PAD140: label is 145;
attribute PINNUM of OUT_PAD141: label is 146;
```

```
attribute PINNUM of OUT_PAD142: label is 147;
attribute PINNUM of OUT_PAD143: label is 148;
attribute PINNUM of OUT_PAD144: label is 149;
attribute PINNUM of OUT_PAD145: label is 150;
attribute PINNUM of OUT_PAD146: label is 151;
attribute PINNUM of OUT_PAD147: label is 152;
attribute PINNUM of OUT_PAD148: label is 153;
attribute PINNUM of OUT_PAD149: label is 154;
attribute PINNUM of OUT_PAD150: label is 155;
attribute PINNUM of OUT_PAD151: label is 156;
attribute PINNUM of OUT_PAD152: label is 157;
attribute PINNUM of OUT_PAD153: label is 158;
attribute PINNUM of OUT_PAD154: label is 159;
attribute PINNUM of OUT_PAD155: label is 160;
attribute PINNUM of OUT_PAD156: label is 181;
attribute PINNUM of OUT_PAD157: label is 182;
attribute PINNUM of OUT_PAD158: label is 183;
attribute PINNUM of OUT_PAD159: label is 184;
attribute PINNUM of OUT_PAD160: label is 185;
attribute PINNUM of OUT_PAD161: label is 186;
attribute PINNUM of OUT_PAD162: label is 187;
attribute PINNUM of OUT_PAD163: label is 188;
attribute PINNUM of OUT_PAD164: label is 189;
attribute PINNUM of OUT_PAD165: label is 190;
attribute PINNUM of OUT_PAD166: label is 191;
attribute PINNUM of OUT_PAD167: label is 192;
attribute PINNUM of OUT_PAD168: label is 193;
attribute PINNUM of OUT_PAD169: label is 194;
attribute PINNUM of OUT_PAD170: label is 195;
attribute PINNUM of OUT_PAD171: label is 196;
attribute PINNUM of OUT_PAD172: label is 197;
attribute PINNUM of OUT_PAD173: label is 198;
attribute PINNUM of OUT_PAD174: label is 199;
attribute PINNUM of OUT_PAD175: label is 200;
attribute PINNUM of OUT_PAD176: label is 221;
attribute PINNUM of OUT_PAD177: label is 222;
attribute PINNUM of OUT_PAD178: label is 223;
attribute PINNUM of OUT_PAD179: label is 224;
attribute PINNUM of OUT_PAD180: label is 225;
attribute PINNUM of OUT_PAD181: label is 226;
attribute PINNUM of OUT_PAD182: label is 227;
attribute PINNUM of OUT_PAD183: label is 228;
attribute PINNUM of OUT_PAD184: label is 229;
attribute PINNUM of OUT_PAD185: label is 230;
attribute PINNUM of OUT_PAD186: label is 231;
attribute PINNUM of OUT_PAD187: label is 232;
```

```
attribute PINNUM of OUT_PAD188: label is 233;
attribute PINNUM of OUT_PAD189: label is 234;
attribute PINNUM of OUT_PAD190: label is 235;
attribute PINNUM of OUT_PAD191: label is 236;
attribute PINNUM of OUT_PAD192: label is 237;
attribute PINNUM of OUT_PAD193: label is 238;
attribute PINNUM of OUT_PAD194: label is 239;
attribute PINNUM of OUT_PAD195: label is 240;
attribute PINNUM of OUT_PAD196: label is 261;
attribute PINNUM of OUT_PAD197: label is 262;
attribute PINNUM of OUT_PAD198: label is 263;
attribute PINNUM of OUT_PAD199: label is 264;
attribute PINNUM of OUT_PAD200: label is 265;
attribute PINNUM of OUT_PAD201: label is 266;
attribute PINNUM of OUT_PAD202: label is 267;
attribute PINNUM of OUT_PAD203: label is 268;
attribute PINNUM of OUT_PAD204: label is 269;
attribute PINNUM of OUT_PAD205: label is 270;
attribute PINNUM of OUT_PAD206: label is 271;
attribute PINNUM of OUT_PAD207: label is 272;
attribute PINNUM of OUT_PAD208: label is 273;
attribute PINNUM of OUT_PAD209: label is 274;
attribute PINNUM of OUT_PAD210: label is 275;
attribute PINNUM of OUT_PAD211: label is 276;
attribute PINNUM of OUT_PAD212: label is 277;
attribute PINNUM of OUT_PAD213: label is 278;
attribute PINNUM of OUT_PAD214: label is 279;
attribute PINNUM of OUT_PAD215: label is 280;
attribute PINNUM of OUT_PAD216: label is 301;
attribute PINNUM of OUT_PAD217: label is 302;
attribute PINNUM of OUT_PAD218: label is 303;
attribute PINNUM of OUT_PAD219: label is 304;
attribute PINNUM of OUT_PAD220: label is 305;
attribute PINNUM of OUT_PAD221: label is 306;
attribute PINNUM of OUT_PAD222: label is 307;
attribute PINNUM of OUT_PAD223: label is 308;
attribute PINNUM of OUT_PAD224: label is 309;
attribute PINNUM of OUT_PAD225: label is 310;
attribute PINNUM of OUT_PAD226: label is 311;
attribute PINNUM of OUT_PAD227: label is 312;
attribute PINNUM of OUT_PAD228: label is 313;
attribute PINNUM of OUT_PAD229: label is 314;
attribute PINNUM of OUT_PAD230: label is 315;
attribute PINNUM of OUT_PAD231: label is 316;
attribute PINNUM of OUT_PAD232: label is 317;
attribute PINNUM of OUT_PAD233: label is 318;
```

attribute PINNUM of OUT_PAD234: label is 319;
attribute PINNUM of OUT_PAD235: label is 320;
attribute PINNUM of OUT_PAD236: label is 341;
attribute PINNUM of OUT_PAD237: label is 342;
attribute PINNUM of OUT_PAD238: label is 343;
attribute PINNUM of OUT_PAD239: label is 344;
attribute PINNUM of OUT_PAD240: label is 345;
attribute PINNUM of OUT_PAD241: label is 346;
attribute PINNUM of OUT_PAD242: label is 347;
attribute PINNUM of OUT_PAD243: label is 348;
attribute PINNUM of OUT_PAD244: label is 349;
attribute PINNUM of OUT_PAD245: label is 350;
attribute PINNUM of OUT_PAD246: label is 351;
attribute PINNUM of OUT_PAD247: label is 352;
attribute PINNUM of OUT_PAD248: label is 353;
attribute PINNUM of OUT_PAD249: label is 354;
attribute PINNUM of OUT_PAD250: label is 355;
attribute PINNUM of OUT_PAD251: label is 356;
attribute PINNUM of OUT_PAD252: label is 357;
attribute PINNUM of OUT_PAD253: label is 358;
attribute PINNUM of OUT_PAD254: label is 359;
attribute PINNUM of OUT_PAD255: label is 360;
attribute PINNUM of OUT_PAD256: label is 381;
attribute PINNUM of OUT_PAD257: label is 382;
attribute PINNUM of OUT_PAD258: label is 383;
attribute PINNUM of OUT_PAD259: label is 384;
attribute PINNUM of OUT_PAD260: label is 385;
attribute PINNUM of OUT_PAD261: label is 386;
attribute PINNUM of OUT_PAD262: label is 387;
attribute PINNUM of OUT_PAD263: label is 388;
attribute PINNUM of OUT_PAD264: label is 389;
attribute PINNUM of OUT_PAD265: label is 390;
attribute PINNUM of OUT_PAD266: label is 391;
attribute PINNUM of OUT_PAD267: label is 392;
attribute PINNUM of OUT_PAD268: label is 393;
attribute PINNUM of OUT_PAD269: label is 394;
attribute PINNUM of OUT_PAD270: label is 395;
attribute PINNUM of OUT_PAD271: label is 396;
attribute PINNUM of OUT_PAD272: label is 397;
attribute PINNUM of OUT_PAD273: label is 378;
attribute PINNUM of OUT_PAD274: label is 379;
attribute PINNUM of OUT_PAD275: label is 380;
attribute PINNUM of OUT_PAD276: label is 398;
attribute PINNUM of OUT_PAD277: label is 399;
attribute PINNUM of OUT_PAD278: label is 400;
attribute PINNUM of OUT_PAD279: label is 401;

```
attribute PINNUM of OUT_PAD280: label is 402;
attribute PINNUM of OUT_PAD281: label is 403;
attribute PINNUM of OUT_PAD282: label is 405;
attribute PINNUM of OUT_PAD283: label is 406;
attribute PINNUM of OUT_PAD284: label is 408;
attribute PINNUM of OUT_PAD285: label is 409;
attribute PINNUM of OUT_PAD286: label is 410;
attribute PINNUM of OUT_PAD287: label is 421;
attribute PINNUM of OUT_PAD288: label is 422;
attribute PINNUM of OUT_PAD289: label is 423;
attribute PINNUM of OUT_PAD290: label is 425;
attribute PINNUM of OUT_PAD291: label is 426;
attribute PINNUM of OUT_PAD292: label is 428;
attribute PINNUM of OUT_PAD293: label is 429;
attribute PINNUM of OUT_PAD294: label is 430;

begin

   VDD_PAD0: ap_vdd_20
                    port map (PADPIN=>VDD0);
   GND_PAD0: ap_gnd_20
                    port map (PADPIN=>GND0);

   IN_PAD0: areapadin
                    port map (PADPIN=>mult_in1_opto(0), padto-
core=>mult_in1_opto_int(0));
   IN_PAD1: areapadin
                    port map (PADPIN=>mult_in1_opto(1), padto-
core=>mult_in1_opto_int(1));
   IN_PAD2: areapadin
                    port map (PADPIN=>mult_in1_opto(2), padto-
core=>mult_in1_opto_int(2));
   IN_PAD3: areapadin
                    port map (PADPIN=>mult_in1_opto(3), padto-
core=>mult_in1_opto_int(3));
   IN_PAD4: areapadin
                    port map (PADPIN=>mult_in1_opto(4), padto-
core=>mult_in1_opto_int(4));
   IN_PAD5: areapadin
                    port map (PADPIN=>mult_in1_opto(5), padto-
core=>mult_in1_opto_int(5));
   IN_PAD6: areapadin
                    port map (PADPIN=>mult_in1_opto(6), padto-
core=>mult_in1_opto_int(6));
   IN_PAD7: areapadin
                    port map (PADPIN=>mult_in1_opto(7), padto-
```

```
core=>mult_in1_opto_int(7));
   IN_PAD8: areapadin
                    port map (PADPIN=>mult_in1_opto(8), padto-
core=>mult_in1_opto_int(8));
   IN_PAD9: areapadin
                    port map (PADPIN=>mult_in1_opto(9), padto-
core=>mult_in1_opto_int(9));
   IN_PAD10: areapadin
                    port map (PADPIN=>mult_in1_opto(10), padto-
core=>mult_in1_opto_int(10));
   IN_PAD11: areapadin
                    port map (PADPIN=>mult_in1_opto(11), padto-
core=>mult_in1_opto_int(11));
   IN_PAD12: areapadin
                    port map (PADPIN=>mult_in1_opto(12), padto-
core=>mult_in1_opto_int(12));
   IN_PAD13: areapadin
                    port map (PADPIN=>mult_in1_opto(13), padto-
core=>mult_in1_opto_int(13));
   IN_PAD14: areapadin
                    port map (PADPIN=>mult_in1_opto(14), padto-
core=>mult_in1_opto_int(14));
   IN_PAD15: areapadin
                    port map (PADPIN=>mult_in1_opto(15), padto-
core=>mult_in1_opto_int(15));
   IN_PAD16: areapadin
                    port map (PADPIN=>mult_in1_opto(16), padto-
core=>mult_in1_opto_int(16));
   IN_PAD17: areapadin
                    port map (PADPIN=>mult_in1_opto(17), padto-
core=>mult_in1_opto_int(17));
   IN_PAD18: areapadin
                    port map (PADPIN=>mult_in1_opto(18), padto-
core=>mult_in1_opto_int(18));
   IN_PAD19: areapadin
                    port map (PADPIN=>mult_in1_opto(19), padto-
core=>mult_in1_opto_int(19));
   IN_PAD20: areapadin
                    port map (PADPIN=>mult_in1_opto(20), padto-
core=>mult_in1_opto_int(20));
   IN_PAD21: areapadin
                    port map (PADPIN=>mult_in1_opto(21), padto-
core=>mult_in1_opto_int(21));
   IN_PAD22: areapadin
                    port map (PADPIN=>mult_in1_opto(22), padto-
core=>mult_in1_opto_int(22));
```

```
   IN_PAD23: areapadin
                    port map (PADPIN=>mult_in1_opto(23), padto-
core=>mult_in1_opto_int(23));
   IN_PAD24: areapadin
                    port map (PADPIN=>mult_in1_opto(24), padto-
core=>mult_in1_opto_int(24));
   IN_PAD25: areapadin
                    port map (PADPIN=>mult_in1_opto(25), padto-
core=>mult_in1_opto_int(25));
   IN_PAD26: areapadin
                    port map (PADPIN=>mult_in1_opto(26), padto-
core=>mult_in1_opto_int(26));
   IN_PAD27: areapadin
                    port map (PADPIN=>mult_in1_opto(27), padto-
core=>mult_in1_opto_int(27));
   IN_PAD28: areapadin
                    port map (PADPIN=>mult_in1_opto(28), padto-
core=>mult_in1_opto_int(28));
   IN_PAD29: areapadin
                    port map (PADPIN=>mult_in1_opto(29), padto-
core=>mult_in1_opto_int(29));
   IN_PAD30: areapadin
                    port map (PADPIN=>mult_in1_opto(30), padto-
core=>mult_in1_opto_int(30));
   IN_PAD31: areapadin
                    port map (PADPIN=>mult_in1_opto(31), padto-
core=>mult_in1_opto_int(31));

   IN_PAD32: areapadin
                    port map (PADPIN=>mult_in2_opto(0), padto-
core=>mult_in2_opto_int(0));
   IN_PAD33: areapadin
                    port map (PADPIN=>mult_in2_opto(1), padto-
core=>mult_in2_opto_int(1));
   IN_PAD34: areapadin
                    port map (PADPIN=>mult_in2_opto(2), padto-
core=>mult_in2_opto_int(2));
   IN_PAD35: areapadin
                    port map (PADPIN=>mult_in2_opto(3), padto-
core=>mult_in2_opto_int(3));
   IN_PAD36: areapadin
                    port map (PADPIN=>mult_in2_opto(4), padto-
core=>mult_in2_opto_int(4));
   IN_PAD37: areapadin
                    port map (PADPIN=>mult_in2_opto(5), padto-
core=>mult_in2_opto_int(5));
```

```
  IN_PAD38: areapadin
                port map (PADPIN=>mult_in2_opto(6), padto-
core=>mult_in2_opto_int(6));
  IN_PAD39: areapadin
                port map (PADPIN=>mult_in2_opto(7), padto-
core=>mult_in2_opto_int(7));
  IN_PAD40: areapadin
                port map (PADPIN=>mult_in2_opto(8), padto-
core=>mult_in2_opto_int(8));
  IN_PAD41: areapadin
                port map (PADPIN=>mult_in2_opto(9), padto-
core=>mult_in2_opto_int(9));
  IN_PAD42: areapadin
                port map (PADPIN=>mult_in2_opto(10), padto-
core=>mult_in2_opto_int(10));
  IN_PAD43: areapadin
                port map (PADPIN=>mult_in2_opto(11), padto-
core=>mult_in2_opto_int(11));

  IN_PAD44: areapadin
                port map (PADPIN=>add_in_opto(0), padtocore=>add_in_opto_int(0));
  IN_PAD45: areapadin
                port map (PADPIN=>add_in_opto(1), padtocore=>add_in_opto_int(1));
  IN_PAD46: areapadin
                port map (PADPIN=>add_in_opto(2), padtocore=>add_in_opto_int(2));
  IN_PAD47: areapadin
                port map (PADPIN=>add_in_opto(3), padtocore=>add_in_opto_int(3));
  IN_PAD48: areapadin
                port map (PADPIN=>add_in_opto(4), padtocore=>add_in_opto_int(4));
  IN_PAD49: areapadin
                port map (PADPIN=>add_in_opto(5), padtocore=>add_in_opto_int(5));
  IN_PAD50: areapadin
                port map (PADPIN=>add_in_opto(6), padtocore=>add_in_opto_int(6));
  IN_PAD51: areapadin
                port map (PADPIN=>add_in_opto(7), padtocore=>add_in_opto_int(7));
  IN_PAD52: areapadin
                port map (PADPIN=>add_in_opto(8), padtocore=>add_in_opto_int(8));
  IN_PAD53: areapadin
                port map (PADPIN=>add_in_opto(9), padtocore=>add_in_opto_int(9));
  IN_PAD54: areapadin
                port map (PADPIN=>add_in_opto(10), padtocore=>add_in_opto_int(10));
  IN_PAD55: areapadin
                port map (PADPIN=>add_in_opto(11), padtocore=>add_in_opto_int(11));
  IN_PAD56: areapadin
                port map (PADPIN=>add_in_opto(12), padtocore=>add_in_opto_int(12));
  IN_PAD57: areapadin
```

```
                port map (PADPIN=>add_in_opto(13), padtocore=>add_in_opto_int(13));
IN_PAD58: areapadin
                port map (PADPIN=>add_in_opto(14), padtocore=>add_in_opto_int(14));
IN_PAD59: areapadin
                port map (PADPIN=>add_in_opto(15), padtocore=>add_in_opto_int(15));
IN_PAD60: areapadin
                port map (PADPIN=>add_in_opto(16), padtocore=>add_in_opto_int(16));
IN_PAD61: areapadin
                port map (PADPIN=>add_in_opto(17), padtocore=>add_in_opto_int(17));
IN_PAD62: areapadin
                port map (PADPIN=>add_in_opto(18), padtocore=>add_in_opto_int(18));
IN_PAD63: areapadin
                port map (PADPIN=>add_in_opto(19), padtocore=>add_in_opto_int(19));
IN_PAD64: areapadin
                port map (PADPIN=>add_in_opto(20), padtocore=>add_in_opto_int(20));
IN_PAD65: areapadin
                port map (PADPIN=>add_in_opto(21), padtocore=>add_in_opto_int(21));
IN_PAD66: areapadin
                port map (PADPIN=>add_in_opto(22), padtocore=>add_in_opto_int(22));
IN_PAD67: areapadin
                port map (PADPIN=>add_in_opto(23), padtocore=>add_in_opto_int(23));
IN_PAD68: areapadin
                port map (PADPIN=>add_in_opto(24), padtocore=>add_in_opto_int(24));
IN_PAD69: areapadin
                port map (PADPIN=>add_in_opto(25), padtocore=>add_in_opto_int(25));
IN_PAD70: areapadin
                port map (PADPIN=>add_in_opto(26), padtocore=>add_in_opto_int(26));
IN_PAD71: areapadin
                port map (PADPIN=>add_in_opto(27), padtocore=>add_in_opto_int(27));
IN_PAD72: areapadin
                port map (PADPIN=>add_in_opto(28), padtocore=>add_in_opto_int(28));
IN_PAD73: areapadin
                port map (PADPIN=>add_in_opto(29), padtocore=>add_in_opto_int(29));
IN_PAD74: areapadin
                port map (PADPIN=>add_in_opto(30), padtocore=>add_in_opto_int(30));
IN_PAD75: areapadin
                port map (PADPIN=>add_in_opto(31), padtocore=>add_in_opto_int(31));
IN_PAD76: areapadin
                port map (PADPIN=>add_in_opto(32), padtocore=>add_in_opto_int(32));
IN_PAD77: areapadin
                port map (PADPIN=>add_in_opto(33), padtocore=>add_in_opto_int(33));
IN_PAD78: areapadin
                port map (PADPIN=>add_in_opto(34), padtocore=>add_in_opto_int(34));
IN_PAD79: areapadin
                port map (PADPIN=>add_in_opto(35), padtocore=>add_in_opto_int(35));
IN_PAD80: areapadin
```

```
        port map (PADPIN=>add_in_opto(36), padtocore=>add_in_opto_int(36));
IN_PAD81: areapadin
        port map (PADPIN=>add_in_opto(37), padtocore=>add_in_opto_int(37));
IN_PAD82: areapadin
        port map (PADPIN=>add_in_opto(38), padtocore=>add_in_opto_int(38));
IN_PAD83: areapadin
        port map (PADPIN=>add_in_opto(39), padtocore=>add_in_opto_int(39));
IN_PAD84: areapadin
        port map (PADPIN=>add_in_opto(40), padtocore=>add_in_opto_int(40));
IN_PAD85: areapadin
        port map (PADPIN=>add_in_opto(41), padtocore=>add_in_opto_int(41));
IN_PAD86: areapadin
        port map (PADPIN=>add_in_opto(42), padtocore=>add_in_opto_int(42));
IN_PAD87: areapadin
        port map (PADPIN=>add_in_opto(43), padtocore=>add_in_opto_int(43));
IN_PAD88: areapadin
        port map (PADPIN=>add_in_opto(44), padtocore=>add_in_opto_int(44));
IN_PAD89: areapadin
        port map (PADPIN=>add_in_opto(45), padtocore=>add_in_opto_int(45));
IN_PAD90: areapadin
        port map (PADPIN=>add_in_opto(46), padtocore=>add_in_opto_int(46));
IN_PAD91: areapadin
        port map (PADPIN=>add_in_opto(47), padtocore=>add_in_opto_int(47));
IN_PAD92: areapadin
        port map (PADPIN=>add_in_opto(48), padtocore=>add_in_opto_int(48));
IN_PAD93: areapadin
        port map (PADPIN=>add_in_opto(49), padtocore=>add_in_opto_int(49));
IN_PAD94: areapadin
        port map (PADPIN=>add_in_opto(50), padtocore=>add_in_opto_int(50));
IN_PAD95: areapadin
        port map (PADPIN=>add_in_opto(51), padtocore=>add_in_opto_int(51));
IN_PAD96: areapadin
        port map (PADPIN=>add_in_opto(52), padtocore=>add_in_opto_int(52));
IN_PAD97: areapadin
        port map (PADPIN=>add_in_opto(53), padtocore=>add_in_opto_int(53));
IN_PAD98: areapadin
        port map (PADPIN=>add_in_opto(54), padtocore=>add_in_opto_int(54));
IN_PAD99: areapadin
        port map (PADPIN=>add_in_opto(55), padtocore=>add_in_opto_int(55));
IN_PAD100: areapadin
        port map (PADPIN=>add_in_opto(56), padtocore=>add_in_opto_int(56));
IN_PAD101: areapadin
        port map (PADPIN=>add_in_opto(57), padtocore=>add_in_opto_int(57));
IN_PAD102: areapadin
        port map (PADPIN=>add_in_opto(58), padtocore=>add_in_opto_int(58));
IN_PAD103: areapadin
```

```
        port map (PADPIN=>add_in_opto(59), padtocore=>add_in_opto_int(59));
IN_PAD104: areapadin
        port map (PADPIN=>add_in_opto(60), padtocore=>add_in_opto_int(60));
IN_PAD105: areapadin
        port map (PADPIN=>add_in_opto(61), padtocore=>add_in_opto_int(61));
IN_PAD106: areapadin
        port map (PADPIN=>add_in_opto(62), padtocore=>add_in_opto_int(62));
IN_PAD107: areapadin
        port map (PADPIN=>add_in_opto(63), padtocore=>add_in_opto_int(63));
IN_PAD108: areapadin
        port map (PADPIN=>add_in_opto(64), padtocore=>add_in_opto_int(64));
IN_PAD109: areapadin
        port map (PADPIN=>add_in_opto(65), padtocore=>add_in_opto_int(65));
IN_PAD110: areapadin
        port map (PADPIN=>add_in_opto(66), padtocore=>add_in_opto_int(66));
IN_PAD111: areapadin
        port map (PADPIN=>add_in_opto(67), padtocore=>add_in_opto_int(67));
IN_PAD112: areapadin
        port map (PADPIN=>add_in_opto(68), padtocore=>add_in_opto_int(68));
IN_PAD113: areapadin
        port map (PADPIN=>add_in_opto(69), padtocore=>add_in_opto_int(69));
IN_PAD114: areapadin
        port map (PADPIN=>add_in_opto(70), padtocore=>add_in_opto_int(70));
IN_PAD115: areapadin
        port map (PADPIN=>add_in_opto(71), padtocore=>add_in_opto_int(71));
IN_PAD116: areapadin
        port map (PADPIN=>add_in_opto(72), padtocore=>add_in_opto_int(72));
IN_PAD117: areapadin
        port map (PADPIN=>add_in_opto(73), padtocore=>add_in_opto_int(73));
IN_PAD118: areapadin
        port map (PADPIN=>add_in_opto(74), padtocore=>add_in_opto_int(74));
IN_PAD119: areapadin
        port map (PADPIN=>add_in_opto(75), padtocore=>add_in_opto_int(75));

IN_PAD120: areapadin
        port map (PADPIN=>clk_opto(0), padtocore=>clk_opto_int(0));

-- Perimeter area pads for electrical input
IN_PAD121: areapadin
        port map (PADPIN=>reset_elec(0), padtocore=>reset_elec_int(0));

IN_PAD122: areapadin
        port map (PADPIN=>add_sel_elec(0), padtocore=>add_sel_elec_int(0));

IN_PAD123: areapadin
        port map (PADPIN=>clk_elec(0), padtocore=>clk_elec_int(0));
```

```
   IN_PAD124: areapadin
                     port map (PADPIN=>mult_in1_elec(0), padto-
core=>mult_in1_elec_int(0));
   IN_PAD125: areapadin
                     port map (PADPIN=>mult_in1_elec(1), padto-
core=>mult_in1_elec_int(1));
   IN_PAD126: areapadin
                     port map (PADPIN=>mult_in1_elec(2), padto-
core=>mult_in1_elec_int(2));
   IN_PAD127: areapadin
                     port map (PADPIN=>mult_in1_elec(3), padto-
core=>mult_in1_elec_int(3));
   IN_PAD128: areapadin
                     port map (PADPIN=>mult_in1_elec(4), padto-
core=>mult_in1_elec_int(4));

   IN_PAD129: areapadin
                     port map (PADPIN=>mult_in2_elec(0), padto-
core=>mult_in2_elec_int(0));
   IN_PAD130: areapadin
                     port map (PADPIN=>mult_in2_elec(1), padto-
core=>mult_in2_elec_int(1));

   IN_PAD131: areapadin
                     port map (PADPIN=>oe_sel_elec(0), padtocore=>oe_sel_elec_int(0));

   IN_PAD132: areapadin
                     port map (PADPIN=>add_in_elec(0), padtocore=>add_in_elec_int(0));
   IN_PAD133: areapadin
                     port map (PADPIN=>add_in_elec(1), padtocore=>add_in_elec_int(1));
   IN_PAD134: areapadin
                     port map (PADPIN=>add_in_elec(2), padtocore=>add_in_elec_int(2));
   IN_PAD135: areapadin
                     port map (PADPIN=>add_in_elec(3), padtocore=>add_in_elec_int(3));
   IN_PAD136: areapadin
                     port map (PADPIN=>add_in_elec(4), padtocore=>add_in_elec_int(4));


   OUT_PAD0: areapadout
                     port map (coretopad=>data_out_opto_int(0), PAD-
PIN=>data_out_opto(0));
   OUT_PAD1: areapadout
                     port map (coretopad=>data_out_opto_int(1), PAD-
PIN=>data_out_opto(1));
   OUT_PAD2: areapadout
```

```
                    port map (coretopad=>data_out_opto_int(2), PAD-
PIN=>data_out_opto(2));
   OUT_PAD3: areapadout
                    port map (coretopad=>data_out_opto_int(3), PAD-
PIN=>data_out_opto(3));
   OUT_PAD4: areapadout
                    port map (coretopad=>data_out_opto_int(4), PAD-
PIN=>data_out_opto(4));
   OUT_PAD5: areapadout
                    port map (coretopad=>data_out_opto_int(5), PAD-
PIN=>data_out_opto(5));
   OUT_PAD6: areapadout
                    port map (coretopad=>data_out_opto_int(6), PAD-
PIN=>data_out_opto(6));
   OUT_PAD7: areapadout
                    port map (coretopad=>data_out_opto_int(7), PAD-
PIN=>data_out_opto(7));
   OUT_PAD8: areapadout
                    port map (coretopad=>data_out_opto_int(8), PAD-
PIN=>data_out_opto(8));
   OUT_PAD9: areapadout
                    port map (coretopad=>data_out_opto_int(9), PAD-
PIN=>data_out_opto(9));
   OUT_PAD10: areapadout
                    port map (coretopad=>data_out_opto_int(10), PAD-
PIN=>data_out_opto(10));
   OUT_PAD11: areapadout
                    port map (coretopad=>data_out_opto_int(11), PAD-
PIN=>data_out_opto(11));
   OUT_PAD12: areapadout
                    port map (coretopad=>data_out_opto_int(12), PAD-
PIN=>data_out_opto(12));
   OUT_PAD13: areapadout
                    port map (coretopad=>data_out_opto_int(13), PAD-
PIN=>data_out_opto(13));
   OUT_PAD14: areapadout
                    port map (coretopad=>data_out_opto_int(14), PAD-
PIN=>data_out_opto(14));
   OUT_PAD15: areapadout
                    port map (coretopad=>data_out_opto_int(15), PAD-
PIN=>data_out_opto(15));
   OUT_PAD16: areapadout
                    port map (coretopad=>data_out_opto_int(16), PAD-
PIN=>data_out_opto(16));
   OUT_PAD17: areapadout
                    port map (coretopad=>data_out_opto_int(17), PAD-
```

```
PIN=>data_out_opto(17));
   OUT_PAD18: areapadout
                   port map (coretopad=>data_out_opto_int(18), PAD-
PIN=>data_out_opto(18));
   OUT_PAD19: areapadout
                   port map (coretopad=>data_out_opto_int(19), PAD-
PIN=>data_out_opto(19));
   OUT_PAD20: areapadout
                   port map (coretopad=>data_out_opto_int(20), PAD-
PIN=>data_out_opto(20));
   OUT_PAD21: areapadout
                   port map (coretopad=>data_out_opto_int(21), PAD-
PIN=>data_out_opto(21));
   OUT_PAD22: areapadout
                   port map (coretopad=>data_out_opto_int(22), PAD-
PIN=>data_out_opto(22));
   OUT_PAD23: areapadout
                   port map (coretopad=>data_out_opto_int(23), PAD-
PIN=>data_out_opto(23));
   OUT_PAD24: areapadout
                   port map (coretopad=>data_out_opto_int(24), PAD-
PIN=>data_out_opto(24));
   OUT_PAD25: areapadout
                   port map (coretopad=>data_out_opto_int(25), PAD-
PIN=>data_out_opto(25));
   OUT_PAD26: areapadout
                   port map (coretopad=>data_out_opto_int(26), PAD-
PIN=>data_out_opto(26));
   OUT_PAD27: areapadout
                   port map (coretopad=>data_out_opto_int(27), PAD-
PIN=>data_out_opto(27));
   OUT_PAD28: areapadout
                   port map (coretopad=>data_out_opto_int(28), PAD-
PIN=>data_out_opto(28));
   OUT_PAD29: areapadout
                   port map (coretopad=>data_out_opto_int(29), PAD-
PIN=>data_out_opto(29));
   OUT_PAD30: areapadout
                   port map (coretopad=>data_out_opto_int(30), PAD-
PIN=>data_out_opto(30));
   OUT_PAD31: areapadout
                   port map (coretopad=>data_out_opto_int(31), PAD-
PIN=>data_out_opto(31));
   OUT_PAD32: areapadout
                   port map (coretopad=>data_out_opto_int(32), PAD-
PIN=>data_out_opto(32));
```

```
   OUT_PAD33: areapadout
                  port map (coretopad=>data_out_opto_int(33), PAD-
PIN=>data_out_opto(33));
   OUT_PAD34: areapadout
                  port map (coretopad=>data_out_opto_int(34), PAD-
PIN=>data_out_opto(34));
   OUT_PAD35: areapadout
                  port map (coretopad=>data_out_opto_int(35), PAD-
PIN=>data_out_opto(35));
   OUT_PAD36: areapadout
                  port map (coretopad=>data_out_opto_int(36), PAD-
PIN=>data_out_opto(36));
   OUT_PAD37: areapadout
                  port map (coretopad=>data_out_opto_int(37), PAD-
PIN=>data_out_opto(37));
   OUT_PAD38: areapadout
                  port map (coretopad=>data_out_opto_int(38), PAD-
PIN=>data_out_opto(38));
   OUT_PAD39: areapadout
                  port map (coretopad=>data_out_opto_int(39), PAD-
PIN=>data_out_opto(39));
   OUT_PAD40: areapadout
                  port map (coretopad=>data_out_opto_int(40), PAD-
PIN=>data_out_opto(40));
   OUT_PAD41: areapadout
                  port map (coretopad=>data_out_opto_int(41), PAD-
PIN=>data_out_opto(41));
   OUT_PAD42: areapadout
                  port map (coretopad=>data_out_opto_int(42), PAD-
PIN=>data_out_opto(42));
   OUT_PAD43: areapadout
                  port map (coretopad=>data_out_opto_int(43), PAD-
PIN=>data_out_opto(43));
   OUT_PAD44: areapadout
                  port map (coretopad=>data_out_opto_int(44), PAD-
PIN=>data_out_opto(44));
   OUT_PAD45: areapadout
                  port map (coretopad=>data_out_opto_int(45), PAD-
PIN=>data_out_opto(45));
   OUT_PAD46: areapadout
                  port map (coretopad=>data_out_opto_int(46), PAD-
PIN=>data_out_opto(46));
   OUT_PAD47: areapadout
                  port map (coretopad=>data_out_opto_int(47), PAD-
PIN=>data_out_opto(47));
   OUT_PAD48: areapadout
```

```
            port map (coretopad=>data_out_opto_int(48), PAD-
PIN=>data_out_opto(48));
  OUT_PAD49: areapadout
            port map (coretopad=>data_out_opto_int(49), PAD-
PIN=>data_out_opto(49));
  OUT_PAD50: areapadout
            port map (coretopad=>data_out_opto_int(50), PAD-
PIN=>data_out_opto(50));
  OUT_PAD51: areapadout
            port map (coretopad=>data_out_opto_int(51), PAD-
PIN=>data_out_opto(51));
  OUT_PAD52: areapadout
            port map (coretopad=>data_out_opto_int(52), PAD-
PIN=>data_out_opto(52));
  OUT_PAD53: areapadout
            port map (coretopad=>data_out_opto_int(53), PAD-
PIN=>data_out_opto(53));
  OUT_PAD54: areapadout
            port map (coretopad=>data_out_opto_int(54), PAD-
PIN=>data_out_opto(54));
  OUT_PAD55: areapadout
            port map (coretopad=>data_out_opto_int(55), PAD-
PIN=>data_out_opto(55));
  OUT_PAD56: areapadout
            port map (coretopad=>data_out_opto_int(56), PAD-
PIN=>data_out_opto(56));
  OUT_PAD57: areapadout
            port map (coretopad=>data_out_opto_int(57), PAD-
PIN=>data_out_opto(57));
  OUT_PAD58: areapadout
            port map (coretopad=>data_out_opto_int(58), PAD-
PIN=>data_out_opto(58));
  OUT_PAD59: areapadout
            port map (coretopad=>data_out_opto_int(59), PAD-
PIN=>data_out_opto(59));
  OUT_PAD60: areapadout
            port map (coretopad=>data_out_opto_int(60), PAD-
PIN=>data_out_opto(60));
  OUT_PAD61: areapadout
            port map (coretopad=>data_out_opto_int(61), PAD-
PIN=>data_out_opto(61));
  OUT_PAD62: areapadout
            port map (coretopad=>data_out_opto_int(62), PAD-
PIN=>data_out_opto(62));
  OUT_PAD63: areapadout
            port map (coretopad=>data_out_opto_int(63), PAD-
```

```
PIN=>data_out_opto(63));
   OUT_PAD64: areapadout
                  port map (coretopad=>data_out_opto_int(64), PAD-
PIN=>data_out_opto(64));
   OUT_PAD65: areapadout
                  port map (coretopad=>data_out_opto_int(65), PAD-
PIN=>data_out_opto(65));
   OUT_PAD66: areapadout
                  port map (coretopad=>data_out_opto_int(66), PAD-
PIN=>data_out_opto(66));
   OUT_PAD67: areapadout
                  port map (coretopad=>data_out_opto_int(67), PAD-
PIN=>data_out_opto(67));
   OUT_PAD68: areapadout
                  port map (coretopad=>data_out_opto_int(68), PAD-
PIN=>data_out_opto(68));
   OUT_PAD69: areapadout
                  port map (coretopad=>data_out_opto_int(69), PAD-
PIN=>data_out_opto(69));
   OUT_PAD70: areapadout
                  port map (coretopad=>data_out_opto_int(70), PAD-
PIN=>data_out_opto(70));
   OUT_PAD71: areapadout
                  port map (coretopad=>data_out_opto_int(71), PAD-
PIN=>data_out_opto(71));
   OUT_PAD72: areapadout
                  port map (coretopad=>data_out_opto_int(72), PAD-
PIN=>data_out_opto(72));
   OUT_PAD73: areapadout
                  port map (coretopad=>data_out_opto_int(73), PAD-
PIN=>data_out_opto(73));
   OUT_PAD74: areapadout
                  port map (coretopad=>data_out_opto_int(74), PAD-
PIN=>data_out_opto(74));
   OUT_PAD75: areapadout
                  port map (coretopad=>data_out_opto_int(75), PAD-
PIN=>data_out_opto(75));


   OUT_PAD76: ap_wire
                  port map (coretopad=>power_short_1_int(0), PAD-
PIN=>power_short_1(0));
   OUT_PAD77: ap_wire
                  port map (coretopad=>power_short_1_int(1), PAD-
PIN=>power_short_1(1));
   OUT_PAD78: ap_wire
```

```
                    port map (coretopad=>power_short_1_int(2), PAD-
PIN=>power_short_1(2));
   OUT_PAD79: ap_wire
                    port map (coretopad=>power_short_1_int(3), PAD-
PIN=>power_short_1(3));
   OUT_PAD80: ap_wire
                    port map (coretopad=>power_short_1_int(4), PAD-
PIN=>power_short_1(4));
   OUT_PAD81: ap_wire
                    port map (coretopad=>power_short_1_int(5), PAD-
PIN=>power_short_1(5));
   OUT_PAD82: ap_wire
                    port map (coretopad=>power_short_1_int(6), PAD-
PIN=>power_short_1(6));
   OUT_PAD83: ap_wire
                    port map (coretopad=>power_short_1_int(7), PAD-
PIN=>power_short_1(7));
   OUT_PAD84: ap_wire
                    port map (coretopad=>power_short_1_int(8), PAD-
PIN=>power_short_1(8));
   OUT_PAD85: ap_wire
                    port map (coretopad=>power_short_1_int(9), PAD-
PIN=>power_short_1(9));
   OUT_PAD86: ap_wire
                    port map (coretopad=>power_short_1_int(10), PAD-
PIN=>power_short_1(10));
   OUT_PAD87: ap_wire
                    port map (coretopad=>power_short_1_int(11), PAD-
PIN=>power_short_1(11));
   OUT_PAD88: ap_wire
                    port map (coretopad=>power_short_1_int(12), PAD-
PIN=>power_short_1(12));
   OUT_PAD89: ap_wire
                    port map (coretopad=>power_short_1_int(13), PAD-
PIN=>power_short_1(13));
   OUT_PAD90: ap_wire
                    port map (coretopad=>power_short_1_int(14), PAD-
PIN=>power_short_1(14));
   OUT_PAD91: ap_wire
                    port map (coretopad=>power_short_1_int(15), PAD-
PIN=>power_short_1(15));
   OUT_PAD92: ap_wire
                    port map (coretopad=>power_short_1_int(16), PAD-
PIN=>power_short_1(16));
   OUT_PAD93: ap_wire
                    port map (coretopad=>power_short_1_int(17), PAD-
```

```
PIN=>power_short_1(17));
   OUT_PAD94: ap_wire
                    port map (coretopad=>power_short_1_int(18), PAD-
PIN=>power_short_1(18));
   OUT_PAD95: areapadout
                    port map (coretopad=>power_short_1_int(19), PAD-
PIN=>power_short_1(19));
   OUT_PAD96: ap_wire
                    port map (coretopad=>power_short_1_int(20), PAD-
PIN=>power_short_1(20));
   OUT_PAD97: ap_wire
                    port map (coretopad=>power_short_1_int(21), PAD-
PIN=>power_short_1(21));
   OUT_PAD98: ap_wire
                    port map (coretopad=>power_short_1_int(22), PAD-
PIN=>power_short_1(22));
   OUT_PAD99: ap_wire
                    port map (coretopad=>power_short_1_int(23), PAD-
PIN=>power_short_1(23));
   OUT_PAD100: ap_wire
                    port map (coretopad=>power_short_1_int(24), PAD-
PIN=>power_short_1(24));
   OUT_PAD101: ap_wire
                    port map (coretopad=>power_short_1_int(25), PAD-
PIN=>power_short_1(25));
   OUT_PAD102: ap_wire
                    port map (coretopad=>power_short_1_int(26), PAD-
PIN=>power_short_1(26));
   OUT_PAD103: ap_wire
                    port map (coretopad=>power_short_1_int(27), PAD-
PIN=>power_short_1(27));
   OUT_PAD104: ap_wire
                    port map (coretopad=>power_short_1_int(28), PAD-
PIN=>power_short_1(28));
   OUT_PAD105: ap_wire
                    port map (coretopad=>power_short_1_int(29), PAD-
PIN=>power_short_1(29));
   OUT_PAD106: ap_wire
                    port map (coretopad=>power_short_1_int(30), PAD-
PIN=>power_short_1(30));
   OUT_PAD107: ap_wire
                    port map (coretopad=>power_short_1_int(31), PAD-
PIN=>power_short_1(31));

   OUT_PAD108: ap_wire
                    port map (coretopad=>power_short_2_int(0), PAD-
```

```
PIN=>power_short_2(0));
   OUT_PAD109: ap_wire
                    port map (coretopad=>power_short_2_int(1), PAD-
PIN=>power_short_2(1));
   OUT_PAD110: ap_wire
                    port map (coretopad=>power_short_2_int(2), PAD-
PIN=>power_short_2(2));
   OUT_PAD111: ap_wire
                    port map (coretopad=>power_short_2_int(3), PAD-
PIN=>power_short_2(3));
   OUT_PAD112: ap_wire
                    port map (coretopad=>power_short_2_int(4), PAD-
PIN=>power_short_2(4));
   OUT_PAD113: ap_wire
                    port map (coretopad=>power_short_2_int(5), PAD-
PIN=>power_short_2(5));
   OUT_PAD114: ap_wire
                    port map (coretopad=>power_short_2_int(6), PAD-
PIN=>power_short_2(6));
   OUT_PAD115: areapadout
                    port map (coretopad=>power_short_2_int(7), PAD-
PIN=>power_short_2(7));
   OUT_PAD116: ap_wire
                    port map (coretopad=>power_short_2_int(8), PAD-
PIN=>power_short_2(8));
   OUT_PAD117: ap_wire
                    port map (coretopad=>power_short_2_int(9), PAD-
PIN=>power_short_2(9));
   OUT_PAD118: ap_wire
                    port map (coretopad=>power_short_2_int(10), PAD-
PIN=>power_short_2(10));
   OUT_PAD119: ap_wire
                    port map (coretopad=>power_short_2_int(11), PAD-
PIN=>power_short_2(11));

   OUT_PAD120: ap_wire
                    port map (coretopad=>power_short_3_int(0), PAD-
PIN=>power_short_3(0));
   OUT_PAD121: ap_wire
                    port map (coretopad=>power_short_3_int(1), PAD-
PIN=>power_short_3(1));
   OUT_PAD122: ap_wire
                    port map (coretopad=>power_short_3_int(2), PAD-
PIN=>power_short_3(2));
   OUT_PAD123: ap_wire
                    port map (coretopad=>power_short_3_int(3), PAD-
```

```
PIN=>power_short_3(3));
   OUT_PAD124: ap_wire
                  port map (coretopad=>power_short_3_int(4), PAD-
PIN=>power_short_3(4));
   OUT_PAD125: ap_wire
                  port map (coretopad=>power_short_3_int(5), PAD-
PIN=>power_short_3(5));
   OUT_PAD126: ap_wire
                  port map (coretopad=>power_short_3_int(6), PAD-
PIN=>power_short_3(6));
   OUT_PAD127: ap_wire
                  port map (coretopad=>power_short_3_int(7), PAD-
PIN=>power_short_3(7));
   OUT_PAD128: ap_wire
                  port map (coretopad=>power_short_3_int(8), PAD-
PIN=>power_short_3(8));
   OUT_PAD129: ap_wire
                  port map (coretopad=>power_short_3_int(9), PAD-
PIN=>power_short_3(9));
   OUT_PAD130: ap_wire
                  port map (coretopad=>power_short_3_int(10), PAD-
PIN=>power_short_3(10));
   OUT_PAD131: ap_wire
                  port map (coretopad=>power_short_3_int(11), PAD-
PIN=>power_short_3(11));
   OUT_PAD132: ap_wire
                  port map (coretopad=>power_short_3_int(12), PAD-
PIN=>power_short_3(12));
   OUT_PAD133: ap_wire
                  port map (coretopad=>power_short_3_int(13), PAD-
PIN=>power_short_3(13));
   OUT_PAD134: ap_wire
                  port map (coretopad=>power_short_3_int(14), PAD-
PIN=>power_short_3(14));
   OUT_PAD135: areapadout
                  port map (coretopad=>power_short_3_int(15), PAD-
PIN=>power_short_3(15));
   OUT_PAD136: ap_wire
                  port map (coretopad=>power_short_3_int(16), PAD-
PIN=>power_short_3(16));
   OUT_PAD137: ap_wire
                  port map (coretopad=>power_short_3_int(17), PAD-
PIN=>power_short_3(17));
   OUT_PAD138: ap_wire
                  port map (coretopad=>power_short_3_int(18), PAD-
PIN=>power_short_3(18));
```

```
   OUT_PAD139: ap_wire
                  port map (coretopad=>power_short_3_int(19), PAD-
PIN=>power_short_3(19));
   OUT_PAD140: ap_wire
                  port map (coretopad=>power_short_3_int(20), PAD-
PIN=>power_short_3(20));
   OUT_PAD141: ap_wire
                  port map (coretopad=>power_short_3_int(21), PAD-
PIN=>power_short_3(21));
   OUT_PAD142: ap_wire
                  port map (coretopad=>power_short_3_int(22), PAD-
PIN=>power_short_3(22));
   OUT_PAD143: ap_wire
                  port map (coretopad=>power_short_3_int(23), PAD-
PIN=>power_short_3(23));
   OUT_PAD144: ap_wire
                  port map (coretopad=>power_short_3_int(24), PAD-
PIN=>power_short_3(24));
   OUT_PAD145: ap_wire
                  port map (coretopad=>power_short_3_int(25), PAD-
PIN=>power_short_3(25));
   OUT_PAD146: ap_wire
                  port map (coretopad=>power_short_3_int(26), PAD-
PIN=>power_short_3(26));
   OUT_PAD147: ap_wire
                  port map (coretopad=>power_short_3_int(27), PAD-
PIN=>power_short_3(27));
   OUT_PAD148: ap_wire
                  port map (coretopad=>power_short_3_int(28), PAD-
PIN=>power_short_3(28));
   OUT_PAD149: ap_wire
                  port map (coretopad=>power_short_3_int(29), PAD-
PIN=>power_short_3(29));
   OUT_PAD150: ap_wire
                  port map (coretopad=>power_short_3_int(30), PAD-
PIN=>power_short_3(30));
   OUT_PAD151: ap_wire
                  port map (coretopad=>power_short_3_int(31), PAD-
PIN=>power_short_3(31));
   OUT_PAD152: ap_wire
                  port map (coretopad=>power_short_3_int(32), PAD-
PIN=>power_short_3(32));
   OUT_PAD153: ap_wire
                  port map (coretopad=>power_short_3_int(33), PAD-
PIN=>power_short_3(33));
   OUT_PAD154: ap_wire
```

```
                    port map (coretopad=>power_short_3_int(34), PAD-
PIN=>power_short_3(34));
   OUT_PAD155: areapadout
                    port map (coretopad=>power_short_3_int(35), PAD-
PIN=>power_short_3(35));
   OUT_PAD156: ap_wire
                    port map (coretopad=>power_short_3_int(36), PAD-
PIN=>power_short_3(36));
   OUT_PAD157: ap_wire
                    port map (coretopad=>power_short_3_int(37), PAD-
PIN=>power_short_3(37));
   OUT_PAD158: ap_wire
                    port map (coretopad=>power_short_3_int(38), PAD-
PIN=>power_short_3(38));
   OUT_PAD159: ap_wire
                    port map (coretopad=>power_short_3_int(39), PAD-
PIN=>power_short_3(39));
   OUT_PAD160: ap_wire
                    port map (coretopad=>power_short_3_int(40), PAD-
PIN=>power_short_3(40));
   OUT_PAD161: ap_wire
                    port map (coretopad=>power_short_3_int(41), PAD-
PIN=>power_short_3(41));
   OUT_PAD162: ap_wire
                    port map (coretopad=>power_short_3_int(42), PAD-
PIN=>power_short_3(42));
   OUT_PAD163: ap_wire
                    port map (coretopad=>power_short_3_int(43), PAD-
PIN=>power_short_3(43));
   OUT_PAD164: ap_wire
                    port map (coretopad=>power_short_3_int(44), PAD-
PIN=>power_short_3(44));
   OUT_PAD165: ap_wire
                    port map (coretopad=>power_short_3_int(45), PAD-
PIN=>power_short_3(45));
   OUT_PAD166: ap_wire
                    port map (coretopad=>power_short_3_int(46), PAD-
PIN=>power_short_3(46));
   OUT_PAD167: ap_wire
                    port map (coretopad=>power_short_3_int(47), PAD-
PIN=>power_short_3(47));
   OUT_PAD168: ap_wire
                    port map (coretopad=>power_short_3_int(48), PAD-
PIN=>power_short_3(48));
   OUT_PAD169: ap_wire
                    port map (coretopad=>power_short_3_int(49), PAD-
```

```
PIN=>power_short_3(49));
   OUT_PAD170: ap_wire
                  port map (coretopad=>power_short_3_int(50), PAD-
PIN=>power_short_3(50));
   OUT_PAD171: ap_wire
                  port map (coretopad=>power_short_3_int(51), PAD-
PIN=>power_short_3(51));
   OUT_PAD172: ap_wire
                  port map (coretopad=>power_short_3_int(52), PAD-
PIN=>power_short_3(52));
   OUT_PAD173: ap_wire
                  port map (coretopad=>power_short_3_int(53), PAD-
PIN=>power_short_3(53));
   OUT_PAD174: ap_wire
                  port map (coretopad=>power_short_3_int(54), PAD-
PIN=>power_short_3(54));
   OUT_PAD175: areapadout
                  port map (coretopad=>power_short_3_int(55), PAD-
PIN=>power_short_3(55));
   OUT_PAD176: ap_wire
                  port map (coretopad=>power_short_3_int(56), PAD-
PIN=>power_short_3(56));
   OUT_PAD177: ap_wire
                  port map (coretopad=>power_short_3_int(57), PAD-
PIN=>power_short_3(57));
   OUT_PAD178: ap_wire
                  port map (coretopad=>power_short_3_int(58), PAD-
PIN=>power_short_3(58));
   OUT_PAD179: ap_wire
                  port map (coretopad=>power_short_3_int(59), PAD-
PIN=>power_short_3(59));
   OUT_PAD180: ap_wire
                  port map (coretopad=>power_short_3_int(60), PAD-
PIN=>power_short_3(60));
   OUT_PAD181: ap_wire
                  port map (coretopad=>power_short_3_int(61), PAD-
PIN=>power_short_3(61));
   OUT_PAD182: ap_wire
                  port map (coretopad=>power_short_3_int(62), PAD-
PIN=>power_short_3(62));
   OUT_PAD183: ap_wire
                  port map (coretopad=>power_short_3_int(63), PAD-
PIN=>power_short_3(63));
   OUT_PAD184: ap_wire
                  port map (coretopad=>power_short_3_int(64), PAD-
PIN=>power_short_3(64));
```

```
   OUT_PAD185: ap_wire
                    port map (coretopad=>power_short_3_int(65), PAD-
PIN=>power_short_3(65));
   OUT_PAD186: ap_wire
                    port map (coretopad=>power_short_3_int(66), PAD-
PIN=>power_short_3(66));
   OUT_PAD187: ap_wire
                    port map (coretopad=>power_short_3_int(67), PAD-
PIN=>power_short_3(67));
   OUT_PAD188: ap_wire
                    port map (coretopad=>power_short_3_int(68), PAD-
PIN=>power_short_3(68));
   OUT_PAD189: ap_wire
                    port map (coretopad=>power_short_3_int(69), PAD-
PIN=>power_short_3(69));
   OUT_PAD190: ap_wire
                    port map (coretopad=>power_short_3_int(70), PAD-
PIN=>power_short_3(70));
   OUT_PAD191: ap_wire
                    port map (coretopad=>power_short_3_int(71), PAD-
PIN=>power_short_3(71));
   OUT_PAD192: ap_wire
                    port map (coretopad=>power_short_3_int(72), PAD-
PIN=>power_short_3(72));
   OUT_PAD193: ap_wire
                    port map (coretopad=>power_short_3_int(73), PAD-
PIN=>power_short_3(73));
   OUT_PAD194: ap_wire
                    port map (coretopad=>power_short_3_int(74), PAD-
PIN=>power_short_3(74));
   OUT_PAD195: areapadout
                    port map (coretopad=>power_short_3_int(75), PAD-
PIN=>power_short_3(75));

   OUT_PAD196: ap_wire
                    port map (coretopad=>power_short_4_int(0), PAD-
PIN=>power_short_4(0));

   OUT_PAD197: ap_wire
                    port map (coretopad=>power_short_5_int(0), PAD-
PIN=>power_short_5(0));
   OUT_PAD198: ap_wire
                    port map (coretopad=>power_short_5_int(1), PAD-
PIN=>power_short_5(1));
   OUT_PAD199: ap_wire
                    port map (coretopad=>power_short_5_int(2), PAD-
```

```
PIN=>power_short_5(2));
   OUT_PAD200: ap_wire
               port map (coretopad=>power_short_5_int(3), PAD-
PIN=>power_short_5(3));
   OUT_PAD201: ap_wire
               port map (coretopad=>power_short_5_int(4), PAD-
PIN=>power_short_5(4));
   OUT_PAD202: ap_wire
               port map (coretopad=>power_short_5_int(5), PAD-
PIN=>power_short_5(5));
   OUT_PAD203: ap_wire
               port map (coretopad=>power_short_5_int(6), PAD-
PIN=>power_short_5(6));
   OUT_PAD204: ap_wire
               port map (coretopad=>power_short_5_int(7), PAD-
PIN=>power_short_5(7));
   OUT_PAD205: ap_wire
               port map (coretopad=>power_short_5_int(8), PAD-
PIN=>power_short_5(8));
   OUT_PAD206: ap_wire
               port map (coretopad=>power_short_5_int(9), PAD-
PIN=>power_short_5(9));
   OUT_PAD207: ap_wire
               port map (coretopad=>power_short_5_int(10), PAD-
PIN=>power_short_5(10));
   OUT_PAD208: ap_wire
               port map (coretopad=>power_short_5_int(11), PAD-
PIN=>power_short_5(11));
   OUT_PAD209: ap_wire
               port map (coretopad=>power_short_5_int(12), PAD-
PIN=>power_short_5(12));
   OUT_PAD210: ap_wire
               port map (coretopad=>power_short_5_int(13), PAD-
PIN=>power_short_5(13));
   OUT_PAD211: ap_wire
               port map (coretopad=>power_short_5_int(14), PAD-
PIN=>power_short_5(14));
   OUT_PAD212: ap_wire
               port map (coretopad=>power_short_5_int(15), PAD-
PIN=>power_short_5(15));
   OUT_PAD213: ap_wire
               port map (coretopad=>power_short_5_int(16), PAD-
PIN=>power_short_5(16));
   OUT_PAD214: ap_wire
               port map (coretopad=>power_short_5_int(17), PAD-
PIN=>power_short_5(17));
```

```
   OUT_PAD215: areapadout
                    port map (coretopad=>power_short_5_int(18), PAD-
PIN=>power_short_5(18));
   OUT_PAD216: ap_wire
                    port map (coretopad=>power_short_5_int(19), PAD-
PIN=>power_short_5(19));
   OUT_PAD217: ap_wire
                    port map (coretopad=>power_short_5_int(20), PAD-
PIN=>power_short_5(20));
   OUT_PAD218: ap_wire
                    port map (coretopad=>power_short_5_int(21), PAD-
PIN=>power_short_5(21));
   OUT_PAD219: ap_wire
                    port map (coretopad=>power_short_5_int(22), PAD-
PIN=>power_short_5(22));
   OUT_PAD220: ap_wire
                    port map (coretopad=>power_short_5_int(23), PAD-
PIN=>power_short_5(23));
   OUT_PAD221: ap_wire
                    port map (coretopad=>power_short_5_int(24), PAD-
PIN=>power_short_5(24));
   OUT_PAD222: ap_wire
                    port map (coretopad=>power_short_5_int(25), PAD-
PIN=>power_short_5(25));
   OUT_PAD223: ap_wire
                    port map (coretopad=>power_short_5_int(26), PAD-
PIN=>power_short_5(26));
   OUT_PAD224: ap_wire
                    port map (coretopad=>power_short_5_int(27), PAD-
PIN=>power_short_5(27));
   OUT_PAD225: ap_wire
                    port map (coretopad=>power_short_5_int(28), PAD-
PIN=>power_short_5(28));
   OUT_PAD226: ap_wire
                    port map (coretopad=>power_short_5_int(29), PAD-
PIN=>power_short_5(29));
   OUT_PAD227: ap_wire
                    port map (coretopad=>power_short_5_int(30), PAD-
PIN=>power_short_5(30));
   OUT_PAD228: ap_wire
                    port map (coretopad=>power_short_5_int(31), PAD-
PIN=>power_short_5(31));
   OUT_PAD229: ap_wire
                    port map (coretopad=>power_short_5_int(32), PAD-
PIN=>power_short_5(32));
   OUT_PAD230: ap_wire
```

```
                    port map (coretopad=>power_short_5_int(33), PAD-
PIN=>power_short_5(33));
   OUT_PAD231: ap_wire
                    port map (coretopad=>power_short_5_int(34), PAD-
PIN=>power_short_5(34));
   OUT_PAD232: ap_wire
                    port map (coretopad=>power_short_5_int(35), PAD-
PIN=>power_short_5(35));
   OUT_PAD233: ap_wire
                    port map (coretopad=>power_short_5_int(36), PAD-
PIN=>power_short_5(36));
   OUT_PAD234: ap_wire
                    port map (coretopad=>power_short_5_int(37), PAD-
PIN=>power_short_5(37));
   OUT_PAD235: areapadout
                    port map (coretopad=>power_short_5_int(38), PAD-
PIN=>power_short_5(38));
   OUT_PAD236: ap_wire
                    port map (coretopad=>power_short_5_int(39), PAD-
PIN=>power_short_5(39));
   OUT_PAD237: ap_wire
                    port map (coretopad=>power_short_5_int(40), PAD-
PIN=>power_short_5(40));
   OUT_PAD238: ap_wire
                    port map (coretopad=>power_short_5_int(41), PAD-
PIN=>power_short_5(41));
   OUT_PAD239: ap_wire
                    port map (coretopad=>power_short_5_int(42), PAD-
PIN=>power_short_5(42));
   OUT_PAD240: ap_wire
                    port map (coretopad=>power_short_5_int(43), PAD-
PIN=>power_short_5(43));
   OUT_PAD241: ap_wire
                    port map (coretopad=>power_short_5_int(44), PAD-
PIN=>power_short_5(44));
   OUT_PAD242: ap_wire
                    port map (coretopad=>power_short_5_int(45), PAD-
PIN=>power_short_5(45));
   OUT_PAD243: ap_wire
                    port map (coretopad=>power_short_5_int(46), PAD-
PIN=>power_short_5(46));
   OUT_PAD244: ap_wire
                    port map (coretopad=>power_short_5_int(47), PAD-
PIN=>power_short_5(47));
   OUT_PAD245: ap_wire
                    port map (coretopad=>power_short_5_int(48), PAD-
```

```
PIN=>power_short_5(48));
   OUT_PAD246: ap_wire
                    port map (coretopad=>power_short_5_int(49), PAD-
PIN=>power_short_5(49));
   OUT_PAD247: ap_wire
                    port map (coretopad=>power_short_5_int(50), PAD-
PIN=>power_short_5(50));
   OUT_PAD248: ap_wire
                    port map (coretopad=>power_short_5_int(51), PAD-
PIN=>power_short_5(51));
   OUT_PAD249: ap_wire
                    port map (coretopad=>power_short_5_int(52), PAD-
PIN=>power_short_5(52));
   OUT_PAD250: ap_wire
                    port map (coretopad=>power_short_5_int(53), PAD-
PIN=>power_short_5(53));
   OUT_PAD251: ap_wire
                    port map (coretopad=>power_short_5_int(54), PAD-
PIN=>power_short_5(54));
   OUT_PAD252: ap_wire
                    port map (coretopad=>power_short_5_int(55), PAD-
PIN=>power_short_5(55));
   OUT_PAD253: ap_wire
                    port map (coretopad=>power_short_5_int(56), PAD-
PIN=>power_short_5(56));
   OUT_PAD254: ap_wire
                    port map (coretopad=>power_short_5_int(57), PAD-
PIN=>power_short_5(57));
   OUT_PAD255: areapadout
                    port map (coretopad=>power_short_5_int(58), PAD-
PIN=>power_short_5(58));
   OUT_PAD256: ap_wire
                    port map (coretopad=>power_short_5_int(59), PAD-
PIN=>power_short_5(59));
   OUT_PAD257: ap_wire
                    port map (coretopad=>power_short_5_int(60), PAD-
PIN=>power_short_5(60));
   OUT_PAD258: ap_wire
                    port map (coretopad=>power_short_5_int(61), PAD-
PIN=>power_short_5(61));
   OUT_PAD259: ap_wire
                    port map (coretopad=>power_short_5_int(62), PAD-
PIN=>power_short_5(62));
   OUT_PAD260: ap_wire
                    port map (coretopad=>power_short_5_int(63), PAD-
PIN=>power_short_5(63));
```

```
   OUT_PAD261: ap_wire
                    port map (coretopad=>power_short_5_int(64), PAD-
PIN=>power_short_5(64));
   OUT_PAD262: ap_wire
                    port map (coretopad=>power_short_5_int(65), PAD-
PIN=>power_short_5(65));
   OUT_PAD263: ap_wire
                    port map (coretopad=>power_short_5_int(66), PAD-
PIN=>power_short_5(66));
   OUT_PAD264: ap_wire
                    port map (coretopad=>power_short_5_int(67), PAD-
PIN=>power_short_5(67));
   OUT_PAD265: ap_wire
                    port map (coretopad=>power_short_5_int(68), PAD-
PIN=>power_short_5(68));
   OUT_PAD266: ap_wire
                    port map (coretopad=>power_short_5_int(69), PAD-
PIN=>power_short_5(69));
   OUT_PAD267: ap_wire
                    port map (coretopad=>power_short_5_int(70), PAD-
PIN=>power_short_5(70));
   OUT_PAD268: ap_wire
                    port map (coretopad=>power_short_5_int(71), PAD-
PIN=>power_short_5(71));
   OUT_PAD269: ap_wire
                    port map (coretopad=>power_short_5_int(72), PAD-
PIN=>power_short_5(72));
   OUT_PAD270: ap_wire
                    port map (coretopad=>power_short_5_int(73), PAD-
PIN=>power_short_5(73));
   OUT_PAD271: ap_wire
                    port map (coretopad=>power_short_5_int(74), PAD-
PIN=>power_short_5(74));
   OUT_PAD272: ap_wire
                    port map (coretopad=>power_short_5_int(75), PAD-
PIN=>power_short_5(75));

   OUT_PAD273: areapadout
                    port map (coretopad=>dummy_int(0), PADPIN=>dummy(0));
   OUT_PAD274: areapadout
                    port map (coretopad=>dummy_int(1), PADPIN=>dummy(1));
   OUT_PAD275: areapadout
                    port map (coretopad=>dummy_int(2), PADPIN=>dummy(2));
   OUT_PAD276: areapadout
                    port map (coretopad=>dummy_int(3), PADPIN=>dummy(3));
   OUT_PAD277: areapadout
```

```
                    port map (coretopad=>dummy_int(4), PADPIN=>dummy(4));
    OUT_PAD278: areapadout
                    port map (coretopad=>dummy_int(5), PADPIN=>dummy(5));

    -- Perimeter area pads for electrical output
    OUT_PAD279: areapadout
                    port map (coretopad=>data_out_elec_int(0), PADPIN=>data_out_elec(0));
    OUT_PAD280: areapadout
                    port map (coretopad=>data_out_elec_int(1), PADPIN=>data_out_elec(1));
    OUT_PAD281: areapadout
                    port map (coretopad=>data_out_elec_int(2), PADPIN=>data_out_elec(2));
    OUT_PAD282: areapadout
                    port map (coretopad=>data_out_elec_int(3), PADPIN=>data_out_elec(3));
    OUT_PAD283: areapadout
                    port map (coretopad=>data_out_elec_int(4), PADPIN=>data_out_elec(4));
    OUT_PAD284: areapadout
                    port map (coretopad=>data_out_elec_int(5), PADPIN=>data_out_elec(5));
    OUT_PAD285: areapadout
                    port map (coretopad=>data_out_elec_int(6), PADPIN=>data_out_elec(6));
    OUT_PAD286: areapadout
                    port map (coretopad=>data_out_elec_int(7), PADPIN=>data_out_elec(7));
    OUT_PAD287: areapadout
                    port map (coretopad=>data_out_elec_int(8), PADPIN=>data_out_elec(8));
    OUT_PAD288: areapadout
                    port map (coretopad=>data_out_elec_int(9), PADPIN=>data_out_elec(9));
    OUT_PAD289: areapadout
                    port map (coretopad=>data_out_elec_int(10), PAD-
PIN=>data_out_elec(10));
    OUT_PAD290: areapadout
                    port map (coretopad=>data_out_elec_int(11), PAD-
PIN=>data_out_elec(11));
    OUT_PAD291: areapadout
                    port map (coretopad=>data_out_elec_int(12), PAD-
PIN=>data_out_elec(12));
    OUT_PAD292: areapadout
                    port map (coretopad=>data_out_elec_int(13), PAD-
PIN=>data_out_elec(13));
    OUT_PAD293: areapadout
                    port map (coretopad=>data_out_elec_int(14), PAD-
PIN=>data_out_elec(14));
    OUT_PAD294: areapadout
                    port map (coretopad=>data_out_elec_int(15), PAD-
PIN=>data_out_elec(15));


    BUFF_mult_in1_opto: buff
```

```
                generic map (N=>32, DPFLAG=>0)
                port map (IN0=>mult_in1_opto_int, Y=>mult_in1_opto_int_buff);
BUFF_mult_in2_opto: buff
                generic map (N=>12, DPFLAG=>0)
                port map (IN0=>mult_in2_opto_int, Y=>mult_in2_opto_int_buff);
BUFF_add_in_opto: buff
                generic map (N=>76, DPFLAG=>0)
                port map (IN0=>add_in_opto_int, Y=>add_in_opto_int_buff);
BUFF_clk_opto: buff
                generic map (N=>1, DPFLAG=>0)
                port map (IN0=>clk_opto_int, Y=>clk_opto_int_buff);

BUFF_mult_in1_elec: buff
                generic map (N=>5, DPFLAG=>0)
                port map (IN0=>mult_in1_elec_int, Y=>mult_in1_elec_int_buff);
BUFF_mult_in2_elec: buff
                generic map (N=>2, DPFLAG=>0)
                port map (IN0=>mult_in2_elec_int, Y=>mult_in2_elec_int_buff);
BUFF_add_in_elec: buff
                generic map (N=>5, DPFLAG=>0)
                port map (IN0=>add_in_elec_int, Y=>add_in_elec_int_buff);
BUFF_clk_elec: buff
                generic map (N=>1, DPFLAG=>0)
                port map (IN0=>clk_elec_int, Y=>clk_elec_int_buff);
BUFF_oe_sel_elec: buff
                generic map (N=>1, DPFLAG=>0)
                port map (IN0=>oe_sel_elec_int, Y=>oe_sel_elec_int_buff);
BUFF_add_sel_elec: buff
                generic map (N=>1, DPFLAG=>0)
                port map (IN0=>add_sel_elec_int, Y=>add_sel_elec_int_buff);
BUFF_reset_elec: buff
                generic map (N=>1, DPFLAG=>0)
                port map (IN0=>reset_elec_int, Y=>reset_elec_int_buff);

BUFF_data_out_opto: buff
                generic map (N=>76, DPFLAG=>0)
                port map (IN0=>data_out_opto_int_buff, Y=>data_out_opto_int);

BUFF_data_out_elec: buff
                generic map (N=>16, DPFLAG=>0)
                port map (IN0=>data_out_elec_int_buff, Y=>data_out_elec_int);

MUX_MULT_IN1_HIGH: mux2
                generic map (N=>27, DPFLAG=>0)
                port map (IN0=>mult_in1_opto_int_buff(31 downto 5),
                                IN1=>GND_27,
```

```
                    S0=>oe_sel_elec_int_buff,
                    Y=>mult_in1_high);

MUX_MULT_IN1_LOW: mux2
          generic map (N=>5, DPFLAG=>0)
          port map (IN0=>mult_in1_opto_int_buff(4 downto 0),
                    IN1=>mult_in1_elec_int_buff,
                    S0=>oe_sel_elec_int_buff,
                    Y=>mult_in1_low);

MUX_MULT_IN2_HIGH: mux2
          generic map (N=>10, DPFLAG=>0)
          port map (IN0=>mult_in2_opto_int_buff(11 downto 2),
                    IN1=>GND_10,
                    S0=>oe_sel_elec_int_buff,
                    Y=>mult_in2_high);

MUX_MULT_IN2_LOW: mux2
          generic map (N=>2, DPFLAG=>0)
          port map (IN0=>mult_in2_opto_int_buff(1 downto 0),
                    IN1=>mult_in2_elec_int_buff,
                    S0=>oe_sel_elec_int_buff,
                    Y=>mult_in2_low);

MUX_ADD_IN1_HIGH: mux2
          generic map (N=>71, DPFLAG=>0)
          port map (IN0=>add_in_opto_int_buff(75 downto 5),
                    IN1=>GND_71,
                    S0=>oe_sel_elec_int_buff,
                    Y=>add_in_high);

MUX_ADD_IN1_LOW: mux2
          generic map (N=>5, DPFLAG=>0)
          port map (IN0=>add_in_opto_int_buff(4 downto 0),
                    IN1=>add_in_elec_int_buff,
                    S0=>oe_sel_elec_int_buff,
                    Y=>add_in_low);

MUX_ADD_IN2: mux2
          generic map (N=>76, DPFLAG=>0)
          port map (IN0(75 downto 5)=>add_in_high,
                    IN0(4 downto 0)=>add_in_low,
                    IN1=>data_out_opto_int_buff,
                    S0=>add_sel_elec_int_buff,
                    Y=>add_in2);
```

```
    MULTIPLY_ADD: mult_32x12_76
            port map (a(31 downto 5)=>mult_in1_high,
                    a(4 downto 0)=>mult_in1_low,
                    b(11 downto 2)=>mult_in2_high,
                    b(1 downto 0)=>mult_in2_low,
                    c=>add_in2,
                    d=>ff_in);

    MUX_CLK_IN: mux2
            generic map (N=>1, DPFLAG=>0)
            port map (IN0=>clk_opto_int_buff,
                    IN1=>clk_elec_int_buff,
                    S0=>oe_sel_elec_int_buff,
                    Y=>clk_in);

    FF_DATA: dff_c
            generic map (N=>76, DPFLAG=>0)
            port map (CLK=>clk_in, CLR=>reset_elec_int_buff,
                    D=>ff_in, Q=>data_out_opto_int_buff);

    GND_27 <= (others=> ‘0’);
    GND_10 <= (others=> ‘0’);
    GND_71 <= (others=> ‘0’);

    data_out_elec_int_buff <= data_out_opto_int_buff(15 downto 0);


end rgr_structural;
```

APPENDIX1:

# APPENDIX2:SOURCE CODE FOR MULTIPLIER AND ADDER

```
-------------------------------------------------------------------------
--
--  Copyright 1997 Richard Rozier
--
--  Entity name: mad_32x12_76
--
--  Purpose: Creates a 32x12 multiplier and a 44+76 adder.
--
--  Author:  Richard Rozier
--
--  Notes: Uses standard cells from synthesis instead of datapath cells.
--
--  Revision History
--              10/20/97  File created.
-------------------------------------------------------------------------

library IEEE;
--library EPOCH_LIB;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity mult_32x12_76 is
port(
            a         : in std_logic_vector(31 downto 0);
            b         : in std_logic_vector(11 downto 0);
            c         : in std_logic_vector(75 downto 0);
            d         : out std_logic_vector(75 downto 0)
   );
end mult_32x12_76;



architecture behavior of mult_32x12_76 is
signal mult_out : std_logic_vector(43 downto 0);

begin

   mult_out <= signed(a) * signed(b);
   d <= signed(mult_out) + signed(c);

end behavior;
```

# APPENDIX3: SOURCE CODE FOR THE VHDL CODE GENERATOR

```
/* File last modified: 10/16/97*/
#include <stdio.h>
#define SIGNALLIMIT 15

main()
{
   int i, j, k, l, inpad[SIGNALLIMIT], outpad[SIGNALLIMIT];
   int inpadtotal, outpadtotal, inputsignum, outputsignum;
   int inpadcounter, outpadcounter, dummynum, dummystarts;
   int rownum, padnum, offset;
   int dummystartnum[SIGNALLIMIT], dummygap[SIGNALLIMIT];
   int inpadnum[SIGNALLIMIT], outpadnum[SIGNALLIMIT];
   int shortnum, shortsig[SIGNALLIMIT], shortpadnum[SIGNALLIMIT];
   char insignal[SIGNALLIMIT][256], outsignal[SIGNALLIMIT][256];

                  /*  VARIABLE DESCRIPTION                 */
                  /*  i, j, k, l = various counters      */
                  /*  inpad, outpad = arrays that hold the number of pads*/
                  /*                          for each signal                  */
                  /*  inputsignum, outputsignum = the number if input and*/
                  /*                                            output signals            */
                  /*  inpadtotal, outpadtotal = the total number of pads*/
                  /*  insignal, outsignal = the names of each signal*/
                  /*  inpadnum, outpadnum = the starting pad number of*/
                  /*                                     each signal                      */
                  /*  shortnum =  the number of shorted signals*/
                  /*  shortpadnum =  the starting pad number of each */
                  /*                          shorted signal                   */
                  /*  shortsig = the number of pads for each signal to */
                  /*               short                                               */
                  /*  dummynum = the total number of dummy pads*/
                  /*  dummystarts = the number of different starting */
                  /*                         points for dummy pads             */
                  /*  dummystartnum = the starting pad number for each*/
                  /*                          set of dummy pads                */
                  /*  dummygap = the number of pads for each group of */
                  /*               dummy pads                                */
                  /*  rownum = the current row number of the pads*/
                  /*  padnum = the current number of the pad*/
                  /*  offset = the column number of the pad*/


   printf("\n\nEnter the number of input signals: ");
```

```
scanf("%d", &inputsignum);
inpadtotal = 0;
for (i=0; i<inputsignum; i++)
{
        printf("\nEnter the name of the input signal #%d: ", i+1);
        scanf("%s", insignal[i]);
        printf("Enter the number of pads for input signal #%d: ", i+1);
        scanf("%d", &inpad[i]);
        inpadtotal = inpadtotal + inpad[i];
        printf("Enter the starting pad number of input signal #%d: ", i+1);
        scanf("%d", &inpadnum[i]);
}
printf("\n\nEnter the number of output signals: ");
scanf("%d", &outputsignum);
outpadtotal = 0;
for (i=0; i<outputsignum; i++)
{
        printf("\nEnter the name of the output signal #%d: ", i+1);
        scanf("%s", outsignal[i]);
        printf("Enter the number of pads for output signal #%d: ", i+1);
        scanf("%d", &outpad[i]);
        outpadtotal = outpadtotal + outpad[i];
        printf("Enter the starting pad number of output signal #%d: ", i+1);
        scanf("%d", &outpadnum[i]);
}
printf("\n\nEnter the number of DUMMY pads: ");
scanf("%d", &dummynum);
dummystarts = 0;
outpadtotal = inpadtotal + 2*outpadtotal + dummynum;
if (dummynum > 0)
{
        printf("Enter the number of different starting pads: ");
        scanf("%d", &dummystarts);
        for (i=0; i<dummystarts; i++)
        {
          printf("Enter the starting number of dummy pad%d: ", i+1);
          scanf("%d", &dummystartnum[i]);
          printf("Enter the number of dummy pads to include: ", i+1);
          scanf("%d", &dummygap[i]);
        }
}
printf("\n\n");

/* CREATE ENTITY PORT DECLARATIONS */
for (i=0; i<inputsignum; i++)
{
```

```
                if (inpad[i] == 1)
                {
                   printf("  %s          : in std_logic;\n", insignal[i], inpad[i]-1);
                }
                else
                {
                   printf("  %s          : in std_logic_vector(%d downto 0);\n", insignal[i],
inpad[i]-1);
                }
   }
   k = 1;
   for (i=0; i<inputsignum; i++)
   {
                if (inpad[i] == 1)
                {
                   printf("  power_short_%d: out std_logic;\n", k++, inpad[i]-1);
                }
                else
                {
                   printf("  power_short_%d: out std_logic_vector(%d downto 0);\n", k++,
inpad[i]-1);
                }
   }
   for (i=0; i<outputsignum; i++)
   {
                if (outpad[i] == 1)
                {
                   printf("  power_short_%d: out std_logic;\n", k++, outpad[i]-1);
                }
                else
                {
                   printf("  power_short_%d: out std_logic_vector(%d downto 0);\n", k++,
outpad[i]-1);
                }
   }
   for (i=0; i<outputsignum; i++)
   {
                if (outpad[i] == 1)
                {
                   printf("  %s          : out std_logic;\n", outsignal[i], outpad[i]-1);
                }
                else
                {
                   printf("  %s          : out std_logic_vector(%d downto 0);\n", outsig-
nal[i], outpad[i]-1);
                }
```

```
    }
    if (dummynum > 0)
                        printf("  dummy      : out std_logic_vector(%d downto 0)\n", dummy-
num-1);

    printf("\n\n");

    /* CREATE INTERNAL SIGNAL DECLARATIONS */
    for (i=0; i<inputsignum; i++)
    {
                    if (inpad[i] == 1)
                    {
                        printf("signal %s_int: std_logic;\n", insignal[i], inpad[i]-1);
                        printf("signal %s_int_buff: std_logic;\n", insignal[i], inpad[i]-1);
                    }
                    else
                    {
                        printf("signal %s_int: std_logic_vector(%d downto 0);\n", insignal[i],
inpad[i]-1);
                        printf("signal %s_int_buff: std_logic_vector(%d downto 0);\n", insig-
nal[i], inpad[i]-1);
                    }
    }
    for (i=0; i<outputsignum; i++)
    {
                    if (outpad[i] == 1)
                    {
                        printf("signal %s_int: std_logic;\n", outsignal[i], outpad[i]-1);
                        printf("signal %s_int_buff: std_logic;\n", outsignal[i], outpad[i]-1);
                    }
                    else
                    {
                        printf("signal %s_int: std_logic_vector(%d downto 0);\n", outsignal[i],
outpad[i]-1);
                        printf("signal %s_int_buff: std_logic_vector(%d downto 0);\n", outsig-
nal[i], outpad[i]-1);
                    }
    }
    k = 1;
    for (i=0; i<inputsignum; i++)
    {
                    if (inpad[i] == 1)
                    {
                        printf("signal power_short_%d_int: std_logic;\n", k++, inpad[i]-1);
                    }
                    else
```

```
            {
              printf("signal power_short_%d_int: std_logic_vector(%d downto 0);\n",
k++, inpad[i]-1);
            }
  }
  for (i=0; i<outputsignum; i++)
  {
            if (outpad[i] == 1)
            {
              printf("signal power_short_%d_int: std_logic;\n", k++, outpad[i]-1);
            }
            else
            {
              printf("signal power_short_%d_int: std_logic_vector(%d downto 0);\n",
k++, outpad[i]-1);
            }
  }
  if (dummynum > 0)
            printf("signal dummy_int: std_logic_vector(%d downto 0);\n", dummy-
num-1);

  printf("\n");

  /* ASSIGN THE PIN NUMBERS TO THE INPUTS */
  inpadcounter = 0;
  for (j=0; j<inputsignum; j++)
  {
            rownum = inpadnum[j]/20;
            offset = inpadnum[j]%20;
            for (i=0; i<inpad[j]; i++)
            {
              padnum = rownum*20 + offset;
              printf("attribute PINNUM of IN_PAD%d: label is %d;\n", inpad-
counter++, padnum);
              if (offset < 20)
                        offset++;
              else
              {
                        offset = 1;
                        rownum = rownum + 2;
              }
            }
  }
  printf("\n");

  /* ASSIGN THE "PAD" ATTRIBUTE TO THE INPUTS */
```

```
   for (i=0; i<inpadtotal; i++)
   {
             printf("attribute PAD of IN_PAD%d: label is \"PAD\";\n", i);
   }

   printf("\n");

   /* ASSIGN THE PIN NUMBERS TO THE OUTPUTS */
   outpadcounter = 0;
   for (j=0; j<outputsignum; j++)
   {
             rownum = outpadnum[j]/20;
             offset = outpadnum[j]%20;
             for (i=0; i<outpad[j]; i++)
             {
               padnum = rownum*20 + offset;
               printf("attribute PINNUM of OUT_PAD%d: label is %d;\n", outpad-
counter++, padnum);
               if (offset < 20)
                      offset++;
               else
               {
                      offset = 1;
                      rownum = rownum + 2;
               }
             }
   }
   printf("\n");

   /* ASSIGN THE "PAD" ATTRIBUTE TO THE OUTPUTS, POWERS, AND DUM-
MIES */
   for (i=0; i<outpadtotal; i++)
   {
             printf("attribute PAD of OUT_PAD%d: label is \"PAD\";\n", i);
   }
   printf("\n");

   /* ASSIGN THE PIN NUMBERS TO THE POWER RAILS */
             /* ASSIGN A POWER PAD (PIN NUMBER) FOR EACH INPUT SIG-
NAL */
   for (j=0; j<inputsignum; j++)
   {
             rownum = inpadnum[j]/20 + 1;
             offset = inpadnum[j]%20;
             for (i=0; i<inpad[j]; i++)
             {
```

```
            padnum = rownum*20 + offset;
            printf("attribute PINNUM of OUT_PAD%d: label is %d;\n", outpad-
counter++, padnum);
            if (offset < 20)
                offset++;
            else
            {
                offset = 1;
                rownum = rownum + 2;
            }
        }
 }
        /* ASSIGN A POWER PAD (PIN NUMBER) FOR EACH OUTPUT SIG-
NAL */
 for (j=0; j<outputsignum; j++)
 {
        rownum = outpadnum[j]/20 + 1;
        offset = outpadnum[j]%20;
        for (i=0; i<outpad[j]; i++)
        {
            padnum = rownum*20 + offset;
            printf("attribute PINNUM of OUT_PAD%d: label is %d;\n", outpad-
counter++, padnum);
            if (offset < 20)
                offset++;
            else
            {
                offset = 1;
                rownum = rownum + 2;
            }
        }
 }

 /* ASSIGN NUMBERS TO THE DUMMY PADS */
 for (j=0; j<dummystarts; j++)
 {
        padnum = dummystartnum[j];
        for (i=0; i<dummygap[j]; i++)
        {
            printf("attribute PINNUM of OUT_PAD%d: label is %d;\n", outpad-
counter++, padnum++);
        }
 }

 printf("\n\n");
```

```
   /* INSTANCE THE INPUT PADS */
   inpadcounter = 0;
   for (j=0; j<inputsignum; j++)
   {
                for (i=0; i<inpad[j]; i++)
                {
                   printf("    IN_PAD%d: areapadin\n", inpadcounter++);
                   printf("  port map (PADPIN=>%s(%d), padtocore=>%s_int(%d));\n",
insignal[j], i, insignal[j], i);
                }
      printf("\n");
   }
   printf("\n");

   /* INSTANCE THE OUTPUT PADS */
   outpadcounter = 0;
   for (j=0; j<outputsignum; j++)
   {
                for (i=0; i<outpad[j]; i++)
                {
                   printf("    OUT_PAD%d: areapadout\n", outpadcounter++);
                   printf("  port map (coretopad=>%s_int(%d), PADPIN=>%s(%d));\n",
outsignal[j], i, outsignal[j], i);
                }
                printf("\n");
   }
   printf("\n");

   /* INSTANCE THE POWER RAIL PADS */
                /* INSTANCE A POWER PAD FOR EVERY INPUT SIGNAL */
   l = 1;
   for (j=0; j<inputsignum; j++)
   {
                k = inpadnum[j];
                for (i=0; i<inpad[j]; i++)
                {
                   if (k%20 != 0)
                   {
                          printf("    OUT_PAD%d: ap_wire\n", outpadcounter++);
                          printf("      port map (coretopad=>power_short_%d_int(%d),
PADPIN=>power_short_%d(%d));\n", l, i, l, i);
                   }
                   else
                   {
                          printf("    OUT_PAD%d: areapadout\n", outpadcounter++);
                          printf("      port map (coretopad=>power_short_%d_int(%d),
```

```
PADPIN=>power_short_%d(%d));\n", l, i, l, i);
                }
                k++;
            }
            l++;
    printf("\n");
  }
            /* INSTANCE A POWER PAD FOR EVERY OUTPUT SIGNAL */
  for (j=0; j<outputsignum; j++)
  {
            k = outpadnum[j];
            for (i=0; i<outpad[j]; i++)
            {
                if (k%20 != 0)
                {
                        printf("    OUT_PAD%d: ap_wire\n", outpadcounter++);
                        printf("      port map (coretopad=>power_short_%d_int(%d),
PADPIN=>power_short_%d(%d));\n", l, i, l, i);
                }
                else
                {
                        printf("    OUT_PAD%d: areapadout\n", outpadcounter++);
                        printf("      port map (coretopad=>power_short_%d_int(%d),
PADPIN=>power_short_%d(%d));\n", l, i, l, i);
                }
                k++;
            }
            l++;
    printf("\n");
  }

  /* INSTANCE OUTPUT PADS FOR THE DUMMY SIGNALS */
  k = 0;
  for (j=0; j<dummystarts; j++)
  {
            for (i=0; i<dummygap[j]; i++)
            {
                printf("    OUT_PAD%d: areapadout\n", outpadcounter++);
                printf("  port map (coretopad=>dummy_int(%d), PAD-
PIN=>dummy(%d));\n", k, k);
                k++;
            }
  }
  printf("\n");

  /* INSTANCE BUFFERS FOR THE INPUTS */
```

```
    for (i=0; i<inputsignum; i++)
    {
                    printf("    BUFF_%s: buff\n", insignal[i]);
                    printf("      generic map (N=>%d, DPFLAG=>0)\n", inpad[i]);
                    printf("      port map (IN0=>%s_int, Y=>%s_int_buff);\n", insignal[i],
insignal[i]);
    }
    printf("\n");

    /* INSTANCE BUFFERS FOR THE OUTPUTS */
    for (i=0; i<outputsignum; i++)
    {
                    printf("    BUFF_%s: buff\n", outsignal[i]);
                    printf("      generic map (N=>%d, DPFLAG=>0)\n", outpad[i]);
                    printf("      port map (IN0=>%s_int_buff, Y=>%s_int);\n", outsignal[i],
outsignal[i]);
    }

}
```